Masterclass .NET: Do Código Gerenciado à Alta Performance

Arquitetura, Performance e Engenharia de Software com C# Moderno.

Instrutor Sênior | 2024

NotebookLM

O Motor do .NET: Portabilidade Binária

Código C#

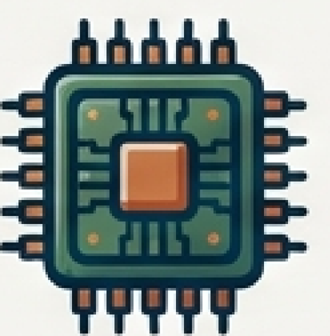
Linguagem expressiva

Compilador

Gera linguagem intermediária

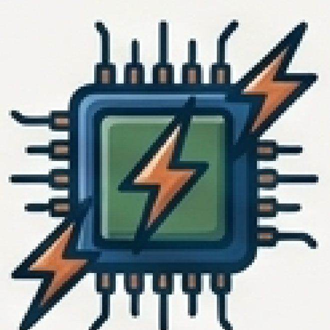
IL (CIL)

Receita universal portátil



Runtime (CLR)

Bifurcação: JIT ou AOT



Código de Máquina

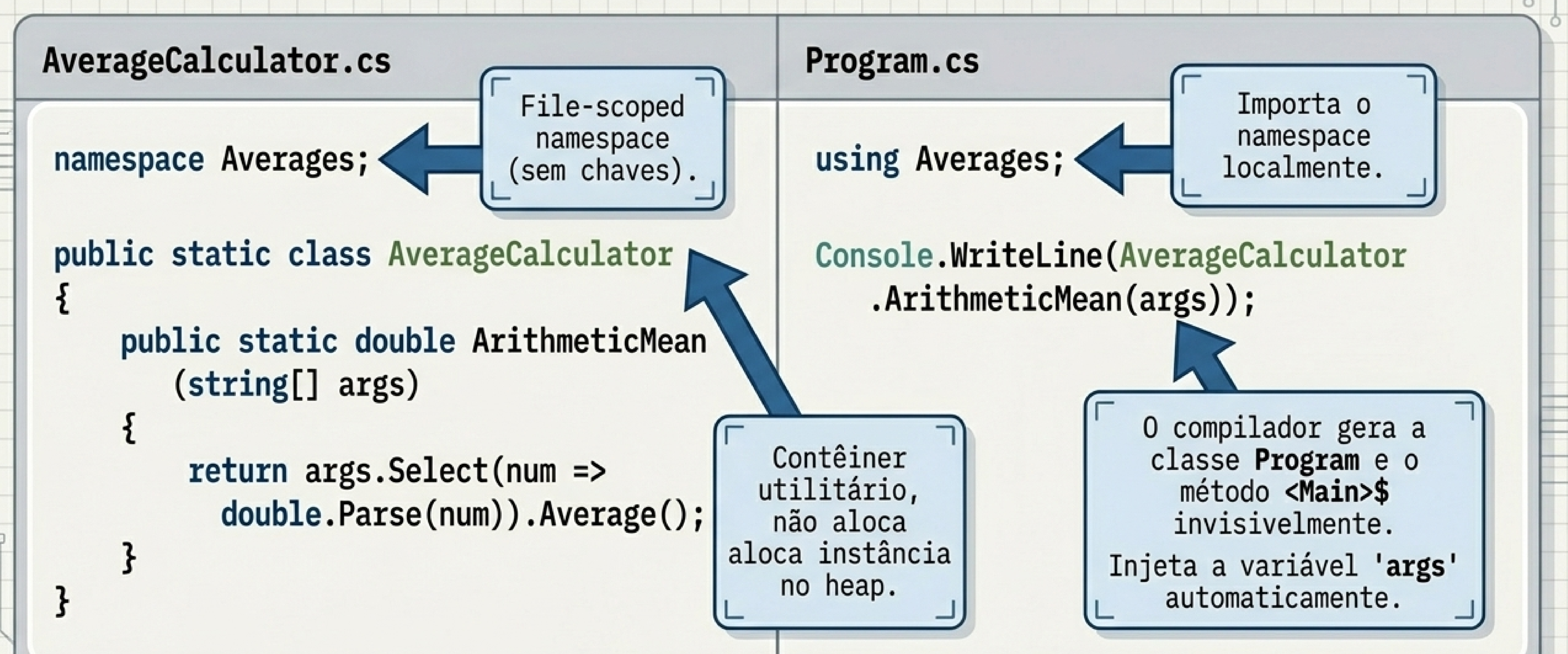
Otimizado para a CPU (x64, ARM64)

IMPLICAÇÃO PRÁTICA: A compilação para IL garante portabilidade. O mesmo Assembly roda em servidores Linux, dispositivos móveis e WebAssembly sem reescrita, preservando a eficiência.

NotebookLM

Estruturação Lógica:
O Fluxo Mental
A menor unidade independente. Gera um Assembly (.dll ou .exe).
1. Problema ->
Projeto (.csproj)
.dll
.exe
O mapa lógico. Informa à IDE que múltiplos projetos andam juntos.
2. Agrupamento ->
Solution (.sln)
Permissão explícita para um projeto enxergar os tipos públicos de outro.
3. Dependências ->
Referências
References
A
B
O "sobrenome" das classes. Previne colisão de nomes na memória.
4. Isolamento ->
Namespaces
Namespace X
Class A
Class B
Namespace Y
Class C
REFORÇO ATIVO
Qual a diferença entre Namespace e Reference?
Resposta: Namespace organiza nomes logicamente; Reference fornece o binário físico para compilação.
NotebookLM

Anatomia de Execução: Top-Level Statements



NotebookLM

Dissecando o LINQ e Lambdas

```
args.Select(numText => double.Parse(numText)).Average();
```

Entrada: Array ["1", "2", "3"]

Select(...) -> Cria um **iterador** (execução *lazy*, sem processamento imediato).

Lambda (numText => double.Parse) -> O CLR aplica a conversão em cada item projetado. Lança FormatException se inválido.

Average() -> Consome o iterador, forçando a avaliação item a item, soma e divide.

Saída: Retorna 2.0 (double)

Nota Arquitetural: Foco em extrema **expressividade** e **legibilidade**, ideal para conjuntos de dados de pequeno a médio porte.

NotebookLM

Anti-Padrões Críticos em C#

O Perigo do "static" Mutável

Problema:
Variáveis estáticas alteráveis (ex: public static List<string> Cache = new();).

```
public static List<string> Cache = new();
```

Impacto:
Não thread-safe. Testes paralelos interferem entre si, gerando falsos positivos. Violação direta da Inversão de Dependência.

Solução:
Use Injeção de Dependência (Singleton) para estado compartilhado. Classes static devem ser estritamente puras e sem estado.

O Custo Oculto da Reflection

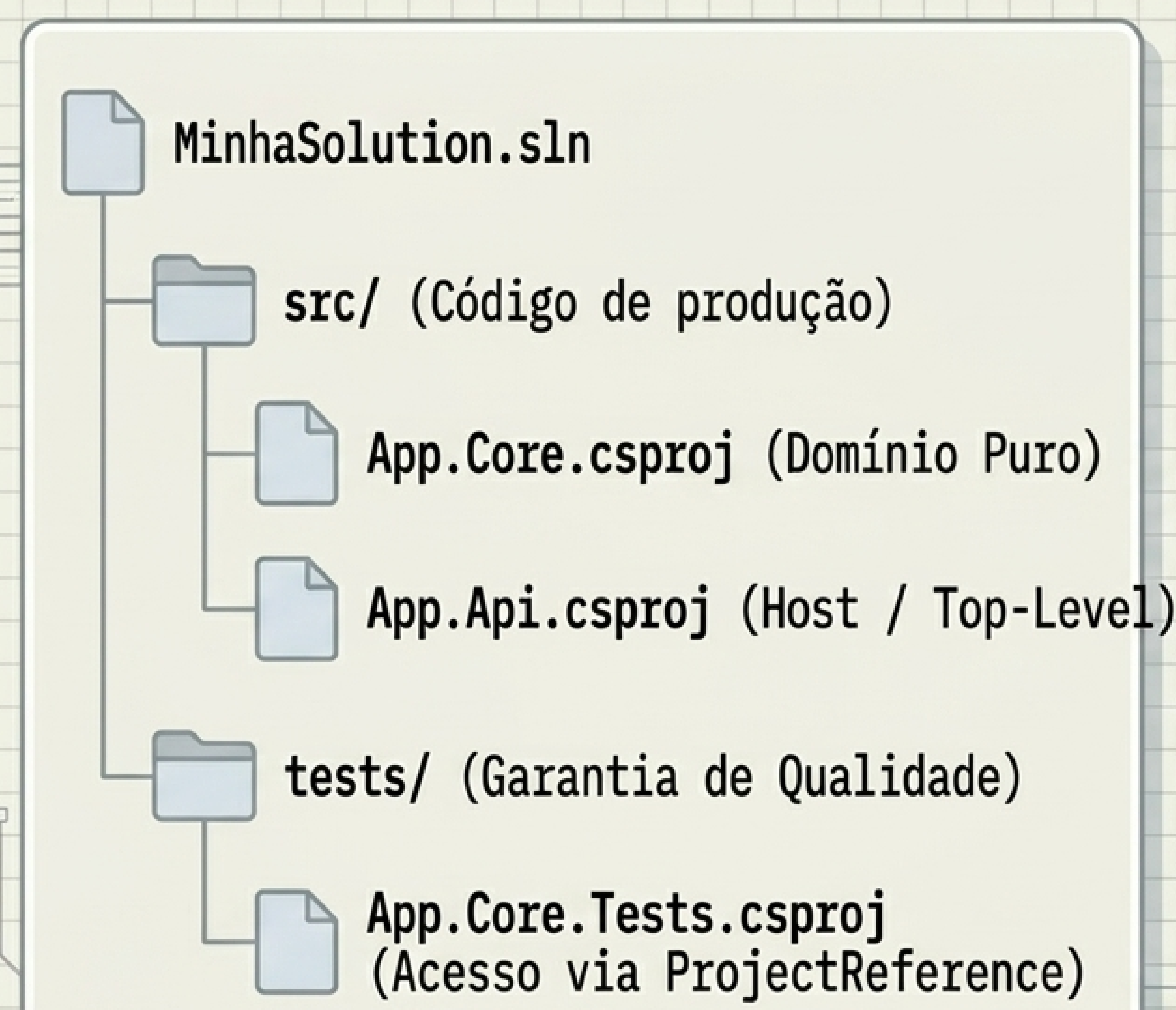
Problema:
Uso abusivo de invocação dinâmica e Assembly.Load.

Impacto:
Chamadas via reflection são ~100x mais lentas. Quebra completamente a capacidade de compilação Native AOT (method trimming).

Solução:
Utilize Source Generators para gerar código em tempo de compilação, mantendo a performance e segurança de tipos.

NotebookLM

Arquitetura Profissional: Estrutura Física e Lógica



Global Usings Estratégico

```
global using System.Text.Json;
global using App.Core.Models;
```

Regra de Ouro: Centralize apenas namespaces usados em mais de 30% dos arquivos do projeto. Importe domínios específicos localmente para manter as dependências explícitas.

NotebookLM

Matriz de Decisão: JIT vs. Native AOT

Dimensão Técnica	JIT (Tiered Compilation Padrão)	Native AOT
Tempo de Inicialização	Mais lento (Compila Tier 0 na 1ª execução)	Quase instantâneo (Cold starts < 50ms)
Tamanho de Deploy	Pequeno (apenas IL) + requer instalação do Runtime	Grande (Binário autossuficiente com runtime embutido)
Otimização de Performance	Dinâmica (Hotspots são inlined em runtime)	Estática (Fixada no momento do build)
Limitações de Código	Nenhuma restrição	Não suporta Reflection.Emit (ex: plugins dinâmicos)
Cenário Ideal de Uso	Web APIs (ASP.NET Core), Serviços de longa duração	CLI Tools, Containers Minimalistas, Serverless

NotebookLM

Engenharia de Performance: Alocação Zero

Abordagem Expressiva (LINQ)

```
args.Select(double.Parse).Average();
```

Diagnóstico: Alta legibilidade.
Custo: Aloca delegates, iteradores e arrays no Heap gerenciado. Gera forte pressão no Garbage Collector (GC) em loops massivos.

Abordagem Máxima Performance (Span<T>)

```
double sum = 0;
foreach(var arg in args.AsSpan()) {
    sum += double.Parse(arg, CultureInfo.InvariantCulture);
}
return sum / args.Length;
```

Diagnóstico: O uso de AsSpan() evita cópias de memória. Sem LINQ significa zero alocação de delegates. **Resultado:** Zero pressão no GC. Mandatório para hot-paths e bibliotecas base.

NotebookLM

Prática Guiada (Fase 1): Modelando o Domínio

1. Imutabilidade com Record

```
public record Venda(string Data,
       string Produto, int Qtd,
       double Valor);
```

Usamos 'record' para imutabilidade nativa, semântica de valor (excelente para testes) e sintaxe ultraconcisa.

2. Princípio Open/Closed

```
public interface IVendaParse
{
   bool PodeProcessar(string cabecalho);
   IEnumerable<Venda> Parse(string[] linhas);
}
```

Garante extensibilidade. Permite adicionar suporte a JSON ou XML no futuro sem alterar o motor principal do sistema.

3. Implementação Concreta

```
public class VendaParserCsv : IVendaParser {
   // Implementação específica para arquivos .csv
}
```

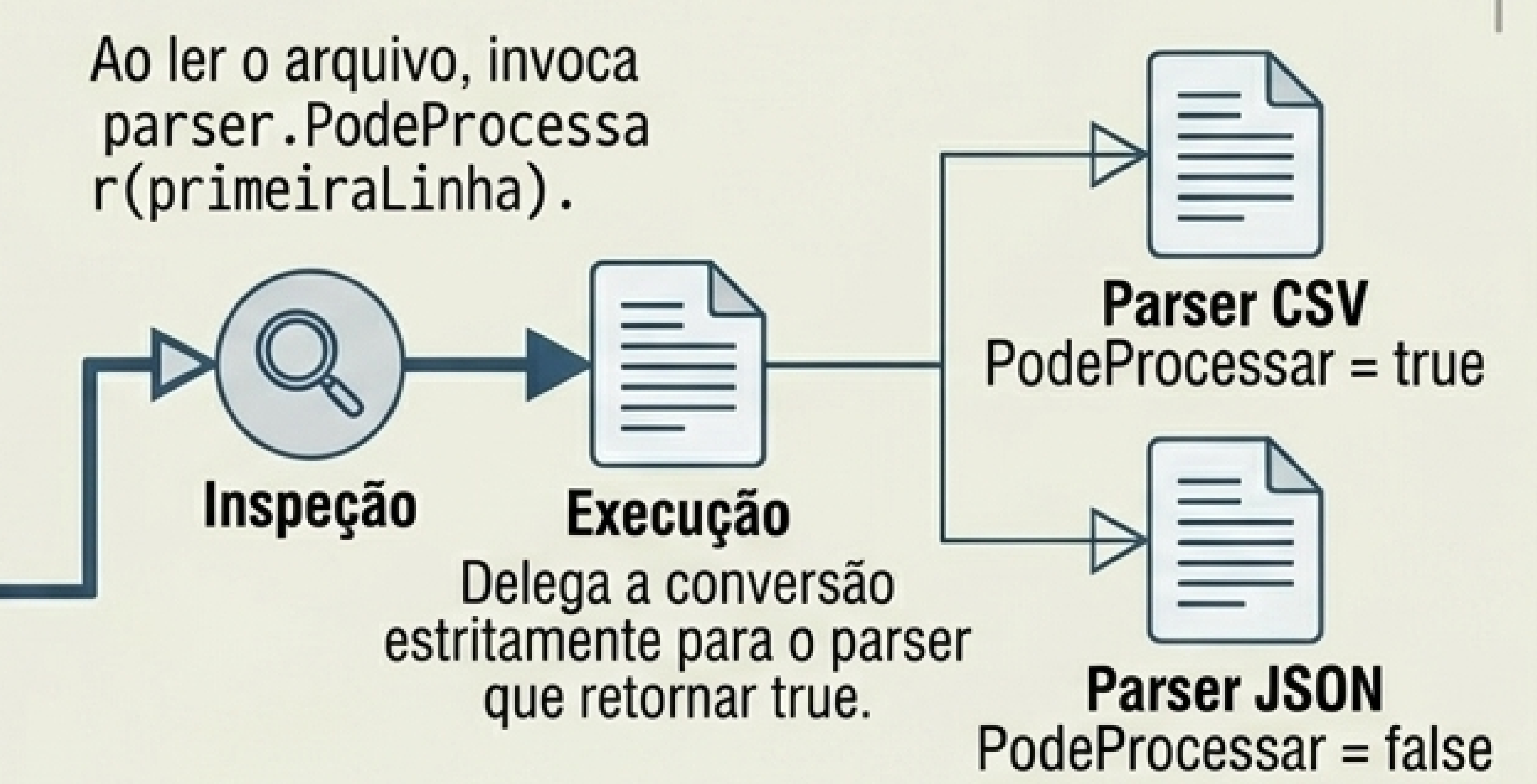
NotebookLM

Prática Guiada (Fase 2): Inversão de Controle

Motor: LeitorVendas

Recebe um array de IVendaParser via construtor.

```
public class LeitorVendas {
    ...
    private readonly IEnumerable<IVendaPars
        er> _parsers;
    ...

    public LeitorVendas(
        IEnumerable<IVendaParser> parsers)
    {
        _parsers = parsers; ...
    }
}
```



Mental Note

E se a linha do CSV estiver corrompida?

Resposta: O parser lança FormatException.
O LeitorVendas captura a exceção, ignora a linha com um aviso no console, e acumula o erro. Isso evita a quebra de um lote inteiro de processamento por causa de uma única linha suja.

NotebookLM

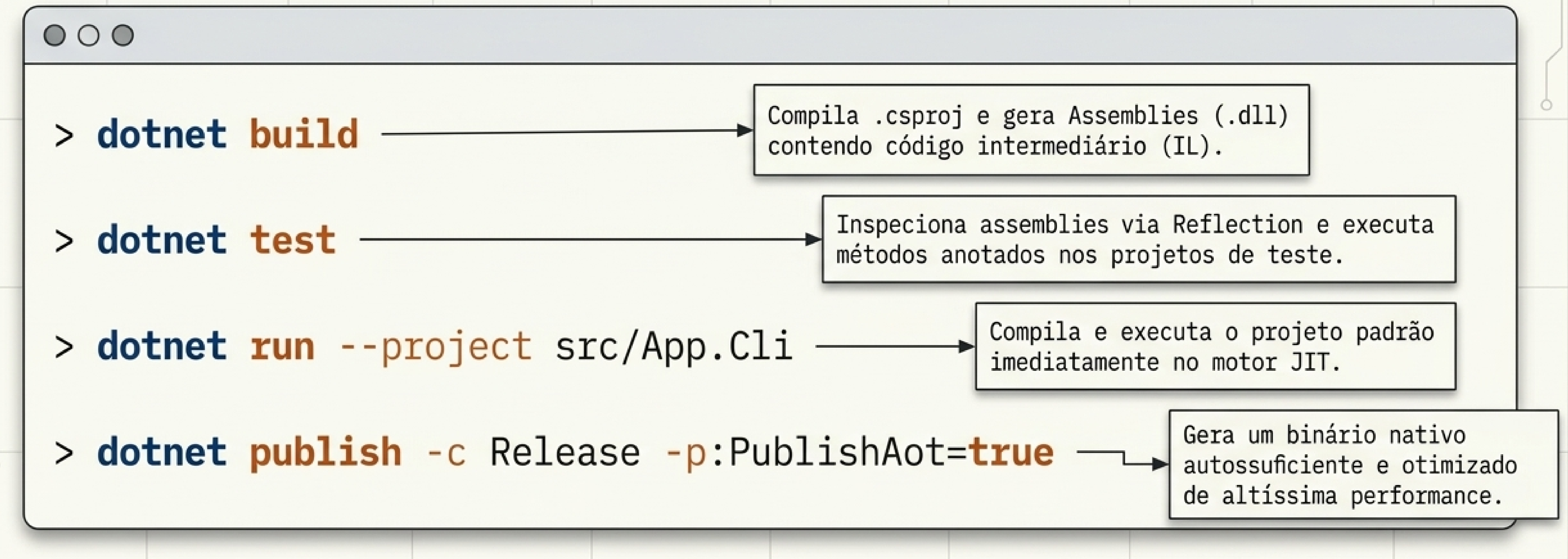
Qualidade: Testes Unitários Padrão AAA

```
[TestMethod]
public void Parse_LinhaValida_RetornaVenda()
{
    1. Arrange (Preparação)
    var parser = new VendaParserCsv();
    var linhas = new[] { "Data,Produto,Qtd,Valor", "2024-01-01,Notebook,1,3500.00" };

    2. Act (Ação)
    var resultado = parser.Parse(linhas).First();

    3. Assert (Validação)
    Assert.AreEqual("Notebook", resultado.Produto);
    Assert.AreEqual(3500.00, resultado.Valor);
}
```

Nomenclatura Rigorosa: [Método]_[Cenário]_[ComportamentoEsperado]. Frameworks usam Reflection em runtime para localizar e executar métodos anotados com [TestMethod] ou [Fact].

NotebookLM

O Ecossistema Operacional: CLI e Deploy



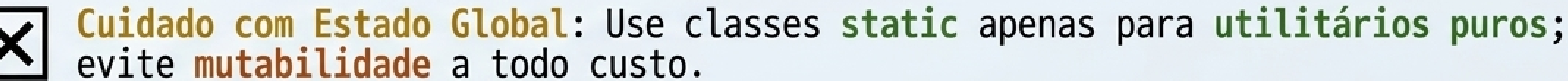
Arquitetura Multi-Stage (Docker): A imagem final gerada via AOT pode ter ~45MB rodando isoladamente em Alpine Linux, eliminando a necessidade do SDK completo no servidor.

NotebookLM

Checklist do Desenvolvedor C# Sênior

Blueprint Checklist

- ☒ **Isole o Domínio**: Separe src/ e tests/. O Core nunca deve depender de interfaces externas ou UI.



- ☒ **Cuidado com Estado Global**: Use classes static apenas para utilitários puros; evite mutabilidade a todo custo.
- ☒ **Imutabilidade por Padrão**: Prefira o uso de record para modelar dados e DTOs, garantindo segurança em concorrência.
- ☒ **Otimize os Hot-Paths**: Substitua LINQ pesado por Span<T> em loops críticos para zerar a pressão no Garbage Collector.
- ☒ **Teste com Intenção**: Aplique o padrão AAA estrito com nomenclaturas descritivas para documentação viva.
- ☒ **Valide sua Arquitetura**: O mesmo código IL deve estar pronto para compilar JIT ou Native AOT conforme a demanda de escalabilidade.

A performance excepcional não é mágica; é o domínio absoluto sobre as engrenagens do runtime.

NotebookLM