Tarefa por Fases — Interfaces em C# (equipes, repositório único, avaliação por equipe)

Introdução (para postar no ClassHero)

Objetivo geral. Consolidar a jornada **procedural → OO → interfaces → repository** com foco em design consciente (ISP, baixo acoplamento, testabilidade). O trabalho será **realizado em equipes e avaliado em equipes**, em **Fases** numeradas e encadeadas. Cada Fase tem um **enunciado** e uma **descrição** com orientações para execução.

Organização por Fases. Todas as entregas serão registradas como **Fase 0, Fase 1, Fase 2, ...** até a Fase final (mini-projeto). O(a) professor(a) fará a **distribuição de equipes para cada Fase** (as equipes podem variar entre fases). Cada Fase possui uma parte teórica, exemplos e uma **tarefa prática** que deve ser entregue no ClassHero com link/arquivo do repositório.

Repositório único (obrigatório). Cada equipe manterá **um repositório único** para todo o trabalho, com **projetos/pastas específicas por Fase**, garantindo rastreabilidade e continuidade.

Estrutura sugerida:

```
repo-raiz/
  README.md
  src/
    fase-00-aquecimento/
    fase-01-procedural/
    fase-02-oo-sem-interface/
    fase-03-com-interfaces/
    fase-04-repository-inmemory/
    fase-05-repository-csv/
    fase-06-repository-json/
    fase-07-isp/
    fase-08-testes-dubles/
    fase-09-cheiros-antidotos/
    fase-10-eixos-opcional/
    fase-11-mini-projeto/
  tests/
    fase-01/ ... (opcional; pode ficar junto de cada projeto)
  docs/
    arquitetura/    # diagramas/resumos curtos por Fase
    decisoes/       # ADRs curtos (opcional)
```

> Observação: o `README.md` na raiz **deve conter a composição da equipe (nomes e RAs)**, o sumário de Fases e instruções de execução/teste por Fase.

README (obrigatório e "vivo"). O `README.md` deve explicar **detalhadamente**: - **Composição da equipe** (nomes completos e RAs) — **item obrigatório**. - **Sumário** com links/âncoras para cada Fase. -

Como executar e testar cada Fase (comandos, projeto alvo, dependências, exemplos de uso). - **Decisões de design** por Fase (1–3 bullets: contratos, ISP, ponto de composição, trocas de implementação). - **Checklist de qualidade** aplicado (contratos coesos, alternância sem alterar cliente, testes sem I/O em unit, ausência de `switch`/downcasts, mudanças pequenas e localizadas). - **Evidências de testes** (prints curtos/logs mínimos quando fizer sentido).

Fluxo de entrega por Fase (simplificado). 1) A equipe cria a **pasta da Fase** no repositório e implementa o solicitado. 2) Atualiza o **README** com composição da equipe, como rodar/testar e as decisões de design da Fase. 3) Publica a entrega no ClassHero (link do repositório + notas da Fase).

Avaliação (por equipe). - **Correção e contrato**: funciona e alterna implementações sem mudar o cliente. - **Qualidade de design**: ISP/coesão, baixo acoplamento, classes folhas `sealed` quando aplicável, nomes claros. - **Testabilidade**: unit sem I/O; dublês adequados; cobertura dos caminhos críticos. - **Clareza de documentação**: README com composição da equipe, decisões de design e instruções de execução/teste. - **Evolução incremental**: mudanças focadas por Fase; histórico coerente e rastreável no repositório único.

Fases (Enunciados + Descrições — sem código)

Fase 0 — Orientações ao aluno (sem código)

Objetivo da Fase
Treinar o olhar de design: identificar **um objetivo fixo** e **peças alternáveis** que realizam esse objetivo por caminhos diferentes. Nomear o **contrato** (o que fazer) e duas **implementações** (como fazer), além de propor uma **política simples** para escolher entre as peças.

Exemplos de situações (para inspirar)
- **Pagamento**: contrato = “processar pagamento”; implementações = Pix / Cartão; política = “valores > R$ 500 → Cartão; senão → Pix”.
- **Envio de notificação**: contrato = “notificar usuário”; implementações = E-mail / SMS; política = “se usuário offline → SMS; se não → E-mail”.
- **Exportação de relatório**: contrato = “exportar relatório”; implementações = CSV / JSON; política = “para integração com planilha → CSV; para API → JSON”.
- **Tradução de texto**: contrato = “traduzir para EN”; implementações = Tradutor humano / API; política = “prazo curto → API; documento crítico → humano”.
- **Compressão de arquivo**: contrato = “comprimir”; implementações = ZIP / GZIP; política = “arquivos múltiplos → ZIP; único e grande → GZIP”.

O que é contrato aqui?
Uma frase curta que descreve **o que** a ação entrega, **sem** falar do “como”. Ex.: “enviar mensagem para o usuário” (não diga “via SMTP”... isso já é implementação).

Como escrever a política (simples e clara)
- **Forma 1 (regra)**: “Se *condição*, use *implementação A*; caso contrário, *implementação B*.”
- **Forma 2 (cenários)**: “No turno noturno → A; em urgência → B.”
- Evite “depende” genérico; sempre **amarre a escolha** a uma condição concreta (tempo, custo, confiabilidade, canal disponível).

Formato da entrega (2 casos reais)
Para **cada** caso, escreva **4–6 linhas**: 1) **Objetivo** (o que se quer alcançar)
2) **Contrato** (o que a peça deve fazer, em 1 linha)
3) **Implementação A** (como 1)
4) **Implementação B** (como 2)
5) **Política** (quando/como escolher A vs. B)
6) **Risco/Observação** (ex.: "SMS tem custo", "CSV perde tipos")

Entregar em texto simples (sem código). Título: "Fase 0 — [Nome da Equipe]".

Critérios de aceite (checklist rápido)
- Cada caso tem **objetivo, contrato, duas implementações e política** bem definida.
- O **contrato** não revela "como" (é descritivo, não técnico).
- As implementações são **alternáveis** (não são variações cosméticas da mesma coisa).
- A política é **concreta e aplicável** (não ambígua).
- Há pelo menos **um risco/observação** por caso.

Erros comuns a evitar
- Confundir contrato com implementação (ex.: "contrato = enviar e-mail").
- Trazer implementações idênticas (ex.: "JSON v1" vs. "JSON v2").
- Política vaga ("depende do contexto" sem dizer **qual** contexto).
- Casos abstratos demais; escolha situações **reais** do cotidiano ou do projeto.

Tempo sugerido
- **Em sala (dinâmico, equipes): 12–15 minutos** total
• 2–3 min para escolher os 2 casos
• 7–9 min para escrever objetivos/contratos/implementações/políticas
• 2–3 min para revisar com o checklist
- **Se quiser aprofundar (aula de 90 min):** estenda para **20 minutos** e peça **1 minuto de pitch** por equipe (apontando o contrato e a política).

Como o professor pode ajudar (perguntas gatilho)
- "O **contrato** está descrevendo o **que** (resultado) ou o **como** (técnica)?"
- "Essas duas implementações são realmente **alternáveis** para o mesmo objetivo?"
- "A **política** é acionável? Dá para uma pessoa terceira aplicá-la sem dúvida?"
- "Qual **risco** existe em cada implementação (custo, latência, erro) e isso muda a política?"

Rubrica enxuta (Fase 0)
- **Clareza do contrato** (0–3)
- **Alternância real entre implementações** (0–3)
- **Política objetiva e aplicável** (0–3)
- **Risco/observação pertinente** (0–1)
Total: 10 pts (sugestão para somar à rubrica geral)

Exemplo de Entrega — Fase 0 (Pagamento)
1) **Objetivo:** permitir que o cliente pague sua compra com sucesso, de forma segura e rápida.
2) **Contrato: processar pagamento** (recebe valor e dados do pagador, retorna sucesso/erro).
3) **Implementação A (como 1): Pix** — transferência instantânea via chave.
4) **Implementação B (como 2): Cartão** — autorização via operadora (crédito/débito).
5) **Política: valores > R$ 500 → Cartão; caso contrário → Pix**.

6) **Risco/Observação:** Pix pode falhar por indisponibilidade do banco; Cartão pode ter taxas/negações; documentar mensagens claras de erro ao cliente.

Fase 1 — Heurística antes do código (mapa mental)

Título para o ClassHero: Fase 1 — Heurística antes do código (mapa mental)

Descrição (para o ClassHero): Em equipes, produzam **uma página de mapa de evolução** para um problema trivial escolhido por vocês (ex.: formatar texto, notificar usuário, calcular frete). O mapa deve mostrar, em três quadros, **como o design evolui**: 1) **Versão procedural** — onde surgem `if/switch` e decisões de modo/variação. 2) **OO sem interface** — quem encapsula o quê; que partes continuam rígidas. 3) **Com interface** — **qual contrato** permite alternar implementações e **onde fica o ponto de composição** (injeção/fábrica). Incluam **3 sinais de alerta previstos** (ex.: interface gorda; downcasts; cliente mudando ao trocar implementação; testes lentos por I/O).

Entregáveis (no repositório único da equipe): - Pasta: `src/fase-01-procedural/` (pode conter apenas o artefato do mapa em `.md` ou imagem `.png/.pdf` nesta fase). - **README.md (raiz) atualizado** com: (a) **composição da equipe (nomes e RAs)**; (b) Sumário das Fases (link/âncora para a Fase 1); (c) onde encontrar e como visualizar o mapa (ex.: `docs/arquitetura/fase-01-mapa.*` ou dentro da pasta da fase).

Formato sugerido do mapa (modelo): - **Problema escolhido (1–2 linhas)** - **Quadro 1 — Procedural:** breve diagrama/fluxo + 2–3 bullets de "onde surgem if/switch". - **Quadro 2 — OO sem interface:** classes envolvidas + 2–3 bullets do que melhorou e do que ficou rígido. - **Quadro 3 — Com interface:** nome do **contrato** e **ponto de composição** + 2–3 bullets do que passa a variar sem tocar o cliente. - **Sinais de alerta (3 itens):** liste os cheiros que você **prevê** se não adotar o contrato/composição.

Critérios de avaliação (rubrica enxuta, 0–10): - **Cobertura dos 3 quadros** (procedural, OO, interface) — 0–3 - **Clareza do contrato e do ponto de composição** — 0–3 - **Qualidade dos 3 sinais de alerta previstos** — 0–2 - **Organização e documentação no README (composição da equipe, links)** — 0–2

Tempo sugerido em sala: 15–20 minutos (equipes) - 3–5 min: escolher o problema trivial - 8–10 min: construir os 3 quadros - 3–5 min: listar sinais de alerta e atualizar o README

Observações / pitfalls a evitar: - Não confundir **contrato** com classe concreta (o contrato descreve "o que", não "como"). - Evitem mapas genéricos: **ancorem em um problema concreto** (frete, formato de saída, notificação). - O **ponto de composição** deve ficar claro (onde as peças são escolhidas), não espalhado no cliente.

Peso sugerido: 10/100 — é a ponte conceitual que guiará as próximas fases com código.

Exemplo preenchido — Fase 1 (tema: Pagamento) - **Problema escolhido:** permitir que o cliente pague a compra escolhendo automaticamente o meio adequado (Pix ou Cartão) conforme regras simples de valor/risco. - **Quadro 1 — Procedural:** fluxo com `if (valor > 500) → Cartão; senão → Pix`; cada novo meio adiciona `if/switch`, aumentando casos e testes. - **Quadro 2 — OO sem interface:** `PixProcessor` e `CardProcessor` encapsulam o "como", mas o serviço ainda decide **qual** concreto usar; cliente muda quando trocamos meio. - **Quadro 3 — Com interface:** contrato = "processar pagamento"; implementações alternáveis; **ponto de composição** lê política "> 500 → Cartão; senão → Pix". **Efeito:** cliente **não muda** ao alternar

peças/política; testes ficam simples com dublês. - **Sinais de alerta (3):** (1) cliente muda ao trocar implementação; (2) ramificações espalhadas; (3) testes lentos/instáveis por I/O real.

Fase 2 — Procedural mínimo (ex.: formatar texto)

Título para o ClassHero: Fase 2 — Procedural mínimo (ex.: formatar texto)

Descrição (para o ClassHero): Em equipes, **implementem a ideia de "modos"** para um objetivo simples (ex.: formatar texto: upper/lower/title; escolher canal: email/sms/push). Devem existir **pelo menos 3 modos + 1 modo padrão**. Entreguem o **fluxo/descrição da função** e **5 cenários de teste/ fronteira descritos em texto**. Expliquem em poucas linhas **por que esse design procedural não escala** (acoplamento a `if/switch`, duplicação, dificuldade de extensão/teste).

Entregáveis (no repositório único da equipe): - Criem a **pasta da Fase 2** no repositório (seguir o padrão já adotado nas fases anteriores). - Um artefato **sem código** (`.md`, `.png` ou `.pdf`) contendo: - **Descrição do objetivo** e dos **modos** escolhidos (mín. 3 + padrão). - **Fluxo da função** (texto ou diagrama simples) destacando onde ocorrem os `if/switch`. - **5 cenários de teste/fronteira** descritos em texto (ver modelo abaixo). - **Notas de limitação**: por que essa abordagem não escala e onde doeria ao adicionar novos modos. - **README.md (raiz)** atualizado com: (a) **composição da equipe (nomes e RAs)**; (b) sumário com link para a Fase 2; (c) onde encontrar o artefato da Fase 2.

Formato sugerido (modelo de entrega em texto): - **Objetivo (1–2 linhas):** o que a função/fluxo pretende entregar. - **Modos (mín. 3 + padrão):** liste os nomes e o efeito esperado de cada modo; indique o **modo padrão**. - **Fluxo procedural (3–6 linhas):** descreva a sequência e **onde** as decisões (`if/switch`) ocorrem. - **5 cenários de teste/fronteira (apenas texto):** 1) valor/entrada mínima (ex.: string vazia) 2) valor/entrada máxima/limite (ex.: tamanho grande) 3) **modo inválido** (esperado cair no **padrão**) 4) combinação que revela ambiguidade entre modos (explicar decisão) 5) caso comum representativo do objetivo - **Por que não escala (4–6 linhas):** riscos claros (ex.: cada modo novo cria mais `if`, espalha decisão, testa combinações, dificulta leitura/manutenção).

Observação importante (C# ao final):

> **Sugestão didática**: ao final desta Fase, **incluir como referência** (no repositório) um pequeno `Program` **em C#** demonstrando a **versão puramente procedural** com `switch/if` (apenas para estudo). **Não é necessário entregar código nesta Fase no ClassHero** — a entrega oficial aqui é **conceitual/textual**.

Critérios de avaliação (rubrica enxuta, 0–10): - **Clareza do objetivo e definição dos modos** (3+) — 0–3 - **Fluxo procedural compreensível** (onde estão as decisões) — 0–3 - **Qualidade dos 5 cenários de teste/fronteira** — 0–2 - **Análise de limites ("por que não escala")** — 0–2

Tempo sugerido em sala: 15–20 minutos - 3–5 min: escolher objetivo e modos - 8–10 min: descrever fluxo e cenários de teste - 3–5 min: redigir limitações e atualizar o README

Pitfalls a evitar: - Modos que não mudam comportamento (variações cosméticas). - "Modo padrão" mal definido (deve ser inequívoco quando nada casar). - Cenários de teste genéricos demais (sem fronteiras/limites/ inválidos). - Crítica superficial ("não escala porque é feio"); foquem em **manutenção/ extensibilidade/testes**.

Peso sugerido: 10/100 — estabelece a dor do procedural e prepara a virada da Fase 3 (polimorfismo/OO sem interface).

Fase 3 — OO sem interface

Título para o ClassHero: Fase 3 — OO sem interface (herança + polimorfismo)

Descrição (para o ClassHero): Transformem a solução da Fase 2 em uma **hierarquia orientada a objetos** com **base comum** e **variações concretas**. A meta é **substituir decisões explícitas** do fluxo por **polimorfismo** (delegar o "como" às subclasses). Mantenham cada classe concreta **restrita à sua responsabilidade** e expliquem **o que melhorou** e **o que ainda ficou rígido** (cliente ainda conhece concretos? ponto de composição ausente?).

Entregáveis (no repositório único da equipe): - Pasta: `src/fase-02-oo-sem-interface/` (mantendo o padrão da estrutura sugerida). - Artefato de design **com pequenos trechos de código C#** (`.md` + snippets) contendo: - **Diagrama/descrição** da hierarquia (base + variações concretas). - **Trechos de código** mostrando a **classe base (virtual/abstract)** e **duas ou mais concretas**. - **Uso do cliente** (instanciação/seleção da concreta) e **onde o** `if/switch` **foi removido** do fluxo central. - **Análise "melhorou vs. rígido"** (2–3 bullets cada). - **README.md (raiz) atualizado** com: (a) **composição da equipe (nomes e RAs)**; (b) Sumário com link para a Fase 3; (c) onde encontrar e como executar o exemplo.

Exemplo mínimo (mesmo domínio da Fase 2: formatar texto)

```
// Base comum: descreve o ritual "formatar" e delega o "como" para as
concretas.
public abstract class TextFormatterBase
{
    public string Format(string text)
    {
        // Passos comuns poderiam ir aqui (ex.: normalização)
        return Apply(text);
    }
    protected abstract string Apply(string text); // passo variável
}

public sealed class UpperCaseFormatter : TextFormatterBase
{
    protected override string Apply(string text) => text?.ToUpperInvariant();
}

public sealed class LowerCaseFormatter : TextFormatterBase
{
    protected override string Apply(string text) => text?.ToLowerInvariant();
}

public sealed class TitleCaseFormatter : TextFormatterBase
{
    protected override string Apply(string text)
    {
```

```
        return System.Text.RegularExpressions.Regex.Replace(text ??
string.Empty,
            "\b(\p{L})", m => m.Value.ToUpperInvariant());
    }
}

// Cliente (ainda conhece concretos): melhora o fluxo, mas mantém a decisão
de composição aqui.
public static class FormatterClient
{
    public static string Render(string text, string mode)
    {
        TextFormatterBase fmt = mode switch
        {
            "UPPER" => new UpperCaseFormatter(),
            "lower" => new LowerCaseFormatter(),
            "Title" => new TitleCaseFormatter(),
            _ => new PassthroughFormatter()
        };
        return fmt.Format(text);
    }
}

public sealed class PassthroughFormatter : TextFormatterBase
{
    protected override string Apply(string text) => text; // padrão: mantém
}
```

> **Observação:** o `switch` acima **saiu do fluxo de formatação** (não há mais ramificações dentro do ritual). Ele permanece **apenas para compor** a concreta inicial. Na Fase 4, essa composição será extraída para um ponto único/contrato.

Como o polimorfismo substitui decisões - Antes: a função procedural **decidia** a cada chamada (vários `if/switch`). - Agora: o **fluxo comum** fica **linear** (classe base); o **passo variável** é **delegado** à concreta via `override`.

Responsabilidades (exemplo) - **TextFormatterBase**: orquestra o ritual (`Format`) e define o gancho `Apply`. - **Upper/Lower/Title/Passthrough**: implementam **somente** o passo variável. - **Cliente**: ainda **escolhe a concreta** (ponto de composição local — será endereçado na Fase 4).

Melhorou - Remoção de `if/switch` no **fluxo central**; leitura mais clara. - **Coesão por variação** (cada concreta trata seu "como"). - **Testes** de cada variação ficam pequenos e focados.

Ainda ficou rígido - **Cliente conhece concretos** (trocar/adição exige mexer no cliente). - **Composição dispersa** (decisão de seleção não é centralizada nem configurável). - **Difícil dobrar** em testes sem um contrato estável (prepara terreno para a Fase 4).

Critérios de avaliação (rubrica enxuta, 0–10) - **Hierarquia clara (base + concretas)** — 0–3 - **Substituição convincente de decisões por polimorfismo** — 0–3 - **Análise "melhorou vs. rígido"** — 0–2 - **README atualizado** (composição da equipe, sumário, execução) — 0–2

Tempo sugerido em sala: 15–20 minutos - 3–5 min: revisar a Fase 2 e definir base + variações - 8–10 min: escrever a base/concretas e o uso do cliente - 3–5 min: redigir análise e atualizar o README

Pitfalls a evitar - Base “faz-tudo” (mantenha apenas o que é realmente comum ao ritual). - Variações **cosméticas** (garanta comportamentos distintos e verificáveis). - Reintroduzir `if/switch` dentro das concretas (o passo variável deve ser simples e específico).

Peso sugerido: 10/100 — consolida a virada de “decidir” → “delegar” (polimorfismo) e prepara a introdução de contratos na próxima fase.

Fase 4 — Interface plugável e testável

Título para o ClassHero: Fase 4 — Interface plugável e testável (contrato + composição + dublês)

Descrição (para o ClassHero): Evoluam a Fase 3 introduzindo um **contrato explícito** para o passo variável e um **ponto único de composição** (política → implementação). O **cliente deve depender do contrato**, não dos concretos. Demonstrem **como testar** o cliente usando **dublês** (fake/stub) **sem I/O**.

Entregáveis (no repositório único da equipe): - Pasta: `src/fase-03-com-interfaces/`. - Artefato em `.md` com trechos de **código C#** mostrando: (1) contrato, (2) duas implementações, (3) cliente dependente do contrato, (4) composição (catálogo/fábrica simples), (5) teste com dublê. - **README.md (raiz)** atualizado com: (a) **composição da equipe (nomes e RAs)**; (b) Sumário com link; (c) como executar os exemplos de uso/teste.

(snippets mantidos conforme versão anterior desta lousa)

Peso sugerido: 10/100 — marco da alternância verdadeira (contrato + composição + testes com dublês).

Fase 5 — Repository (InMemory) — contrato único para acesso a dados

Título para o ClassHero: Fase 5 — Repository InMemory (contrato + implementação em coleção)

Descrição (para o ClassHero): Introduzam o **padrão Repository** como **ponto único** de acesso a dados da equipe (domínio simples, ex.: *Book*, *Student*, *Order*). Entreguem um **contrato genérico** (ou específico do agregado escolhido) e uma **implementação InMemory** baseada em coleção. O **cliente** deve falar **apenas com o Repository**, não com coleções diretamente. Incluam **testes unitários** sem I/O (inserir/listar/buscar/atualizar/remover, com cenários de fronteira).

Entregáveis (no repositório único da equipe): - Pasta: `src/fase-04-repository-inmemory/`. - Arquivo `.md` com diagrama leve e **snippets C#** do contrato + implementação + uso em cliente. - **Testes unitários** cobrindo operações básicas e casos de erro (ex.: id não encontrado). - **README.md (raiz)** atualizado: **composição da equipe (nomes e RAs)**, sumário com link, como executar os testes.

Snippets C# (modelo mínimo)

```
public interface IRepository<T, TId>
{
```

```
    T Add(T entity);
    T? GetById(TId id);
    IReadOnlyList<T> ListAll();
    bool Update(T entity);
    bool Remove(TId id);
}

public sealed class InMemoryRepository<T, TId> : IRepository<T, TId>
{
    private readonly Dictionary<TId, T> _store = new();
    private readonly Func<T, TId> _getId;
    public InMemoryRepository(Func<T, TId> getId) => _getId = getId;

    public T Add(T e) { _store[_getId(e)] = e; return e; }
    public T? GetById(TId id) => _store.TryGetValue(id, out var e) ? e :
default;
    public IReadOnlyList<T> ListAll() => _store.Values.ToList();
    public bool Update(T e) { var id = _getId(e); if (!
_store.ContainsKey(id)) return false; _store[id] = e; return true; }
    public bool Remove(TId id) => _store.Remove(id);
}

// Exemplo de domínio simples
public sealed record Book(int Id, string Title, string Author);

// Uso pelo cliente (sem conhecer coleção)
public static class BookService
{
    public static Book Register(IRepository<Book, int> repo, Book b) =>
repo.Add(b);
}
```

Testes (exemplo xUnit, em texto ou snippet): - Add/ListAll retorna 1 item após inserção. - GetById para id existente retorna entidade; para id ausente retorna null. - Update devolve true quando existe; false caso contrário. - Remove devolve true quando remove; false caso contrário. - Não acessar diretamente coleções no cliente (apenas via Repository).

Critérios de avaliação (rubrica enxuta, 0–10): - **Contrato do Repository** claro e coeso — 0–3 - **Implementação InMemory** correta (coleção, sem I/O) — 0–3 - **Cliente depende só do Repository** — 0–2 - **Testes unitários** cobrindo operações e fronteiras — 0–2

Tempo sugerido em sala: 20–25 minutos - 5–8 min: escolher domínio e modelar contrato - 7–10 min: implementar InMemory + serviço cliente - 5–7 min: escrever testes e atualizar README

Pitfalls a evitar: - Colocar **regra de negócio** dentro do Repository (mantenha-o como **acesso a dados**). - Expor coleções mutáveis (retorne `IReadOnlyList<T>`). - Misturar **política de id** (decidir se o id vem de fora ou é gerado — documentar a escolha).

Peso sugerido: 12/100 — estabelece o **ponto único de dados** e prepara para persistências reais (CSV/JSON) nas próximas fases.

Fase 6 — Repository CSV — persistência em arquivo

Título para o ClassHero: Fase 6 — Repository CSV (persistência em arquivo)

Descrição (para o ClassHero): Evoluam o Repository para **persistir em CSV** (ex.: domínio *Book*). Implementem **Add/GetById/ListAll/Update/Remove** gravando em **arquivo CSV** com **cabeçalho**. Definam claramente a **política de** `Id` (vem de fora ou é gerado) e o **formato CSV** (separador `,`, escape de aspas `"` → `""`, `UTF-8`). Incluam **testes de integração** usando **arquivo temporário** (sem reusar o do colega) e cenários com **vírgulas/aspas** nos campos.

Entregáveis (repo da equipe) - Pasta: `src/fase-05-repository-csv/`. - `.md` com diagrama leve e **snippets C#** do contrato (reaproveitado) + implementação CSV + uso por um serviço cliente. - **Testes** cobrindo: arquivo ausente (criação na primeira escrita), arquivo vazio (lista vazia), inserção, atualização, remoção, leitura com campos contendo vírgulas/aspas, id inexistente. - **README (raiz)** com **composição da equipe (nomes e RAs)**, sumário com link e como executar os testes.

Snippets C# (modelo mínimo)

```
using System.Text;

public sealed class CsvBookRepository : IRepository<Book, int>
{
    private readonly string _path;
    public CsvBookRepository(string path) => _path = path;

    public Book Add(Book e)
    {
        var list = Load();
        list.RemoveAll(b => b.Id == e.Id);
        list.Add(e);
        Save(list);
        return e;
    }

    public Book? GetById(int id) => Load().FirstOrDefault(b => b.Id == id);
    public IReadOnlyList<Book> ListAll() => Load();

    public bool Update(Book e)
    {
        var list = Load();
        var idx = list.FindIndex(b => b.Id == e.Id);
        if (idx < 0) return false;
        list[idx] = e; Save(list); return true;
    }

    public bool Remove(int id)
    {
        var list = Load();
        var removed = list.RemoveAll(b => b.Id == id) > 0;
```

```
        if (removed) Save(list);
        return removed;
    }

    private List<Book> Load()
    {
        if (!File.Exists(_path)) return new();
        var lines = File.ReadAllLines(_path, Encoding.UTF8).ToList();
        if (lines.Count == 0) return new();
        if (lines[0].StartsWith("Id,")) lines.RemoveAt(0); // cabeçalho
        var list = new List<Book>();
        foreach (var line in lines)
        {
            if (string.IsNullOrWhiteSpace(line)) continue;
            var cols = ParseCsv(line);
            var id = int.Parse(cols[0]);
            var title = cols[1];
            var author = cols[2];
            list.Add(new Book(id, title, author));
        }
        return list;
    }

    private void Save(List<Book> list)
    {
        var sb = new StringBuilder();
        sb.AppendLine("Id,Title,Author");
        foreach (var b in list)
            sb.AppendLine(string.Join(',', ToCsv(b.Id.ToString()),
ToCsv(b.Title), ToCsv(b.Author)));
        File.WriteAllText(_path, sb.ToString(), Encoding.UTF8);
    }

    private static string ToCsv(string s)
    {
        if (s is null) return "";
        var needsQuote = s.Contains(',') || s.Contains('"') || s.Contains('
');
        var esc = s.Replace("\"", "\"\"");
        return needsQuote ? $"\"{esc}\"" : esc;
    }

    private static List<string> ParseCsv(string line)
    {
        var result = new List<string>();
        var cur = new StringBuilder();
        bool quoted = false;
        for (int i = 0; i < line.Length; i++)
        {
            var c = line[i];
            if (quoted)
```

```
            {
                if (c == '"')
                {
                    if (i + 1 < line.Length && line[i + 1] == '"') {
cur.Append('"'); i++; }
                    else { quoted = false; }
                }
                else cur.Append(c);
            }
            else
            {
                if (c == ',') { result.Add(cur.ToString()); cur.Clear(); }
                else if (c == '"') quoted = true;
                else cur.Append(c);
            }
        }
        result.Add(cur.ToString());
        return result;
    }
}
```

> **Nota:** implementação didática (não lida com concorrência nem esquemas evolutivos). Em projetos reais, considere bibliotecas especializadas.

Testes (ideia rápida) - Usa `Path.GetTempFileName()` / diretório temporário por teste; limpa ao final. - Verifica criação do cabeçalho na primeira gravação. - Verifica leitura/escrita com `,` e `"` nos campos.

Critérios (0–10) - Persistência CSV correta (inclui cabeçalho/escape) — 0–3 - Operações Add/Get/List/Update/Remove — 0–3 - Testes de integração com temp file — 0–2 - README e documentação de formato/política de id — 0–2

Tempo sugerido: 25–30 minutos

Pitfalls - Não escapar `"` → arquivo inválido.
- Ignorar encoding (usar **UTF-8**).
- Misturar regra de negócio no Repository.

Peso sugerido: 12/100 — consolida persistência simples em arquivo e disciplina de I/O.

Fase 7 — Repository JSON — persistência em JSON

Título para o ClassHero: Fase 7 — Repository JSON (System.Text.Json)

Descrição (para o ClassHero): Evoluam o Repository para **persistir em JSON** (ex.: `books.json`) usando **System.Text.Json**. Implementem as mesmas operações (Add/Get/List/Update/Remove), mantendo **cliente dependente apenas do Repository**. Tratem **arquivo ausente/vazio**, e definam

opções de serialização (**camelCase**, ignorar `null`). Incluam **testes de integração** com arquivo temporário.

Entregáveis (repo da equipe) - Pasta: `src/fase-06-repository-json/`. - `.md` com diagrama leve e **snippets C#** do Repository JSON + uso por serviço cliente. - **Testes** com temp file/diretório, cobrindo operações e arquivo inexistente/vazio. - **README (raiz)** atualizado: **composição da equipe (nomes e RAs)**, sumário e como executar os testes.

Snippets C# (modelo mínimo)

```
using System.Text.Json;
using System.Text.Json.Serialization;

public sealed class JsonBookRepository : IRepository<Book, int>
{
    private readonly string _path;
    private static readonly JsonSerializerOptions _opts = new()
    {
        PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
        DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull,
        WriteIndented = true
    };

    public JsonBookRepository(string path) => _path = path;

    public Book Add(Book e) { var list = Load(); list.RemoveAll(b => b.Id ==
e.Id); list.Add(e); Save(list); return e; }
    public Book? GetById(int id) => Load().FirstOrDefault(b => b.Id == id);
    public IReadOnlyList<Book> ListAll() => Load();
    public bool Update(Book e) { var list = Load(); var i = list.FindIndex(b
=> b.Id == e.Id); if (i < 0) return false; list[i] = e; Save(list); return
true; }
    public bool Remove(int id) { var list = Load(); var ok =
list.RemoveAll(b => b.Id == id) > 0; if (ok) Save(list); return ok; }

    private List<Book> Load()
    {
        if (!File.Exists(_path)) return new();
        var json = File.ReadAllText(_path);
        if (string.IsNullOrWhiteSpace(json)) return new();
        return JsonSerializer.Deserialize<List<Book>>(json, _opts) ?? new();
    }

    private void Save(List<Book> list)
    {
        var json = JsonSerializer.Serialize(list, _opts);
        File.WriteAllText(_path, json);
    }
}
```

Testes (ideia rápida) - `ListAll()` com arquivo inexistente → vazio. - `Add()` cria arquivo e persiste; `GetById()` encontra. - `Update()` retorna `false` se id não existe; `true` quando atualiza. - `Remove()` exclui item e persiste lista.

Critérios (0–10) - Persistência JSON correta (System.Text.Json + opções) — 0–3 - Operações Add/Get/List/Update/Remove — 0–3 - Testes de integração com temp file — 0–2 - README e documentação de formato/decisões — 0–2

Tempo sugerido: 25–30 minutos

Pitfalls - Sobrescrever arquivo sem flush/fechamento (use `File.WriteAllText`). - Opções inconsistentes de serialização entre fases (documente no README). - Misturar regra de negócio no Repository.

Peso sugerido: 12/100 — consolida persistência estruturada, preparando refatorações (mappers, validações e políticas).

Fase 8 — ISP (Interface Segregation Principle) — contratos coesos e mínimos

Título para o ClassHero: Fase 8 — ISP (contratos coesos e mínimos)

Descrição (para o ClassHero): Identifiquem **interfaces gordas** (com métodos que certos clientes não usam) e **segreguem** em **contratos pequenos por capacidade**. Cada cliente deve **depender apenas** do que realmente usa. Ajustem serviços, composição e **testes com dublês mínimos** (sem I/O). Registrem uma **nota de design** explicando a segregação escolhida e seus efeitos.

Entregáveis (repo da equipe) - Pasta: `src/fase-07-isp/`. - `.md` com: (a) diagnóstico da interface gorda (quem usa o quê), (b) contratos segregados, (c) pontos de uso refatorados, (d) dublês mínimos e resultados de teste. - **Snippets C#** mostrando antes → depois (pequenos diffs) e os novos contratos/consumidores. - **README (raiz)** atualizado: **composição da equipe (nomes e RAs)**, sumário com link e como executar os testes.

Snippets C# (modelo mínimo)

```
// Antes: um contrato "gordo" para leitura+escrita
public interface IRepository<T, TId>
{
    T Add(T entity);
    T? GetById(TId id);
    IReadOnlyList<T> ListAll();
    bool Update(T entity);
    bool Remove(TId id);
}

// Depois: contratos mínimos por capacidade (ISP)
public interface IReadRepository<T, TId>
{
    T? GetById(TId id);
```

```
    IReadOnlyList<T> ListAll();
}

public interface IWriteRepository<T, TId>
{
    T Add(T entity);
    bool Update(T entity);
    bool Remove(TId id);
}

// Consumidor que só lê passa a depender SÓ de leitura
public sealed class CatalogQuery
{
    private readonly IReadRepository<Book, int> _read;
    public CatalogQuery(IReadRepository<Book, int> read) => _read = read;
    public Book? FindById(int id) => _read.GetById(id);
    public IReadOnlyList<Book> All() => _read.ListAll();
}

// Dublê mínimo (somente leitura)
file sealed class ReadOnlyFake : IReadRepository<Book, int>
{
    private readonly Dictionary<int, Book> _db = new() { [1] = new(1, "DDD",
"Evans") };
    public Book? GetById(int id) => _db.TryGetValue(id, out var b) ? b :
null;
    public IReadOnlyList<Book> ListAll() => _db.Values.ToList();
}
```

Passos sugeridos 1) **Mapear clientes** e assinalar **quais métodos** usam de cada interface. 2) Propor **segregação** (ex.: `IReadRepository` / `IWriteRepository`; ou `IReader` / `IWriter` para `IStorage`). 3) **Refatorar** consumidores para os contratos mínimos; **ajustar composição**. 4) Atualizar **testes** criando dublês do **tamanho certo** (sem métodos inúteis).

Critérios de avaliação (rubrica enxuta, 0–10) - Diagnóstico claro da interface gorda (quem usa o quê) — 0–2 - Contratos segregados, coesos e nomeados corretamente — 0–3 - Consumidores refatorados dependem só do necessário — 0–3 - Dublês mínimos e testes verdes — 0–2

Tempo sugerido: 20–25 minutos - 5–7 min: mapear usos e desenhar segregação - 7–10 min: refatorar consumidores/composição - 5–8 min: ajustar dublês e registrar nota de design

Pitfalls - Segregar por **camadas** (UI/infra) em vez de **capacidades** (read/write). - Manter métodos “só por compatibilidade” — remova do contrato, crie **adaptadores** se necessário. - Criar muitos contratos quase idênticos — prefira **interfaces estáveis** e **adaptação local**.

Peso sugerido: 10/100 — reduz acoplamento, simplifica dublês e acelera testes, preparando manutenção segura nas próximas etapas.

Fase 9 — Dublês avançados e testes assíncronos (async/stream/tempo)

Título para o ClassHero: Fase 9 — Dublês avançados e testes assíncronos (async, streams e tempo controlado)

Descrição (para o ClassHero): Fortaleçam o design orientado a **costuras**: extraiam dependências externas para **contratos mínimos** (tempo, geração de IDs, leitura/escrita assíncrona) e escrevam **testes unitários** que cobrem **sucesso, erro e cancelamento**, inclusive para `IAsyncEnumerable<T>`. O objetivo é testar **sem I/O real**, com **dublês previsíveis** (fakes/stubs) e **políticas** (retentativa/backoff) controladas.

Entregáveis (repo da equipe) - Pasta: `src/fase-08-dubles-async/`. - `.md` com breve diagrama/explicação e **snippets C#**: 1) Contratos: `IClock`, `IIdGenerator`, `IAsyncReader<T>`, `IAsyncWriter<T>` (ou equivalentes por domínio); 2) Um **serviço cliente** que consome esses contratos (com `async` / `await` e opcionalmente `IAsyncEnumerable<T>`); 3) **Testes unitários** com dublês cobrindo: sucesso, erro, **cancelamento (`CancellationToken`)** e **stream** (curto, vazio, erro no meio); 4) Demonstração de **retentativa/backoff** baseada em política (sem `Thread.Sleep` real; usar clock fake). - **README (raiz)** atualizado: **composição da equipe (nomes e RAs)**, sumário com link, como executar os testes.

Snippets C# (modelo mínimo)

```
public interface IClock { DateTimeOffset Now { get; } }
public interface IIdGenerator { string NewId(); }
public interface IAsyncReader<T> { IAsyncEnumerable<T>
ReadAsync(CancellationToken ct = default); }
public interface IAsyncWriter<T> { Task WriteAsync(T item, CancellationToken
ct = default); }

// Serviço exemplo: lê itens de um reader e escreve em um writer, com
cancelamento e retentativa simples
public sealed class PumpService<T>
{
    private readonly IAsyncReader<T> _reader;
    private readonly IAsyncWriter<T> _writer;
    private readonly IClock _clock;
    public PumpService(IAsyncReader<T> reader, IAsyncWriter<T> writer,
IClock clock)
        => (_reader, _writer, _clock) = (reader, writer, clock);

    public async Task<int> RunAsync(CancellationToken ct)
    {
        var count = 0;
        await foreach (var item in
_reader.ReadAsync(ct).WithCancellation(ct))
        {
            var attempt = 0;
            while (true)
            {
                ct.ThrowIfCancellationRequested();
```

```
                try { await _writer.WriteAsync(item, ct); count++; break; }
                catch when (++attempt <= 3) { /* backoff simulado por clock
fake em teste */ }
            }
        }
        return count;
    }
}
```

Testes com dublês (ideias rápidas) - **Reader fake** que produz 3 itens → `RunAsync` retorna 3. - **Writer stub** que falha 2x e depois escreve → verifica retentativa sem `Sleep` real (controlada por clock fake ou contador). - **Cancelamento**: `CancellationTokenSource.Cancel()` após 1 item → `OperationCanceledException` e contagem parcial. - **Stream vazio**: reader emite 0 itens → retorna 0 sem erro. - **Erro no meio do stream**: reader lança exceção no 2º item → teste verifica propagação/registro.

Critérios de avaliação (rubrica enxuta, 0–10) - Contratos mínimos para costuras (clock/id/async IO) — 0–3 - Serviço cliente assíncrono testável (inclui `IAsyncEnumerable<T>`) — 0–3 - Testes cobrindo sucesso/erro/cancelamento/stream — 0–2 - README claro e decisões justificadas — 0–2

Tempo sugerido: 25–30 minutos - 5–7 min: extrair contratos mínimos (clock/id/async IO) - 10–12 min: implementar serviço cliente e cenários - 8–11 min: escrever testes com dublês e atualizar README

Pitfalls - Usar `Thread.Sleep` em teste (torna lento/não determinístico) — prefira clock fake. - Misturar I/O real (arquivos/rede) — usar dublês em memória. - Não testar **cancelamento** e **erro no meio de stream**.

Peso sugerido: 10/100 — consolida testes assíncronos realistas, reforça costuras e políticas previsíveis.

Fase 10 — Cheiros e antídotos (refatorações com diffs pequenos)

Título para o ClassHero: Fase 10 — Cheiros e antídotos (refatorações guiadas por princípios)

Descrição (para o ClassHero): Identifiquem **cheiros de código/deseño** recorrentes no que construímos e apliquem **antídotos** com **refatorações pequenas** (sem reescrever tudo). O foco é **mudar o ponto certo** e **provar por teste** que o comportamento permaneceu. Entregar uma **coleção de diffs mínimos** (antes → depois) com breve justificativa de design.

Entregáveis (repo da equipe) - Pasta: `src/fase-09-cheiros-antidotos/`. - `.md` com **5 a 7 cheiros** (mínimo 5) contendo, para cada um: (1) **descrição do cheiro**, (2) **antes (snippet curto)**, (3) **depois (snippet curto)**, (4) **antídoto aplicado** e **princípio** associado (ex.: ISP, DIP, SRP), (5) **teste** que demonstra o efeito/segurança. - **README (raiz)** atualizado: **composição da equipe (nomes e RAs)**, sumário com link, como executar testes.

Cheiros sugeridos (escolham 5–7) 1) **Downcast/`as` recorrente no cliente** → *Antídoto:* **contrato específico** + **polimorfismo** (ou visitação controlada).
2) **Interface gorda** (muitos métodos para poucos clientes) → *Antídoto:* **ISP** (segregar por capacidades).
3) **Contrato frágil** (método faz coisas demais) → *Antídoto:* **métodos menores**/contratos focados; **Policy**

Object quando houver muitos parâmetros.
4) **Explosão de subclasses** (variações cosméticas) → *Antídoto:* **estratégia** + **parâmetros/policies**; consolidar comportamento que só muda dado.
5) **Decisão espalhada** (`if/switch` em vários lugares) → *Antídoto:* **ponto único de composição/catálogo**.
6) **Testes lentos com I/O** → *Antídoto:* **dublês** e **costuras** (clock/id/storage) + mover I/O para bordas.
7) **Long Parameter List** → *Antídoto:* **Value Object/Policy Object** e **método de fábrica**.

Exemplos de diffs (C#) — curtos e cirúrgicos

(1) Downcast no cliente → Contrato específico Antes:

```
void Render(object fmt, string s)
{
    if (fmt is UpperCaseFormatter u) Console.WriteLine(u.Apply(s));
    else if (fmt is LowerCaseFormatter l) Console.WriteLine(l.Apply(s));
}
```

Depois (cliente depende do contrato):

```
void Render(ITextFormatter fmt, string s) => Console.WriteLine(fmt.Apply(s));
```

Antídoto: **DIP + Polimorfismo**. Prova: teste injeta fake `ITextFormatter`.

(2) Interface gorda → ISP Antes:

```
public interface IRepository<T, TId>
{ T Add(T e); T? GetById(TId id); IReadOnlyList<T> ListAll(); bool Update(T
e); bool Remove(TId id); }
```

Depois:

```
public interface IReadRepository<T, TId> { T? GetById(TId id);
IReadOnlyList<T> ListAll(); }
public interface IWriteRepository<T, TId> { T Add(T e); bool Update(T e);
bool Remove(TId id); }
```

Antídoto: **ISP**. Prova: consumidor de leitura compila sem write.

(3) Decisão espalhada → Catálogo único Antes:

```
// Cada serviço escolhe o modo de um jeito
if (mode=="UPPER") ...; else if (mode=="lower") ...;
```

Depois:

```
public static class FormatterCatalog
{ public static ITextFormatter Resolve(string m) => m switch{ "UPPER"=>new
UpperCaseFormatter(), "lower"=>new LowerCaseFormatter(), _=>new
PassthroughFormatter() }; }
```

Antídoto: **Composição centralizada**. Prova: testes só variam política.

(4) Long Parameter List → Policy Object Antes:

```
string Export(string path, bool zip, int level, bool async, string mode,
string locale) { ... }
```

Depois:

```
public sealed record ExportPolicy(bool Zip,int Level,bool Async,string
Mode,string Locale);
string Export(string path, ExportPolicy policy) { ... }
```

Antídoto: **VO/Policy Object**. Prova: testes trocam política sem refatorar chamadas.

(5) Teste com I/O → Dublê + costura Antes:

```
var repo = new CsvBookRepository("books.csv"); // usa disco em unit test
```

Depois:

```
var repo = new InMemoryRepository<Book,int>(b=>b.Id); // unit sem I/O
```

Antídoto: **Seams + Dublês**. Prova: testes mais rápidos e determinísticos.

Critérios de avaliação (rubrica enxuta, 0–10) - Escolha pertinente de **5–7 cheiros** (relevância no contexto) — 0–2
- **Antídotos corretos** e alinhados a princípios (ISP/DIP/SRP/OCP) — 0–3
- **Diffs pequenos e claros** (antes → depois) + **teste** que prova segurança — 0–3
- README e justificativas objetivas (o que mudou e por quê) — 0–2

Tempo sugerido: 20–25 minutos - 5–7 min: mapear cheiros no próprio código - 8–10 min: aplicar refatorações mínimas - 5–8 min: ajustar testes e registrar justificativas

Pitfalls - “Refatoração grande-bang” (fugir do objetivo de **pequenas mudanças seguras**).
- Trocar cheiro por outro (ex.: remover downcast e criar interface gorda).
- Não acompanhar com **teste** — cada antídoto precisa de prova.

Peso sugerido: 10/100 — consolida qualidade contínua via melhorias pequenas, seguras e guiadas por princípios.

Fase 11 — Mini-projeto de consolidação (1 sprint curto)

Título para o ClassHero: Fase 11 — Mini-projeto de consolidação (contratos + composição + Repository + ISP + testes)

Descrição (para o ClassHero): Em equipes, construam um **mini-projeto completo** consolidando tudo o que foi praticado: **procedural→OO→interfaces**, **composição/políticas**, **Repository** (InMemory + persistência real **CSV ou JSON**), **ISP** (contratos coesos), e **testes com dublês** (inclui cenários assíncronos quando aplicável). O escopo deve ser pequeno mas **funcional**, com **fluxos de uso reais** e **README detalhado**.

Domínio sugerido (escolham 1): - **Catálogo de Livros** (Book) com cadastro, busca por título/autor, atualização e remoção; exportar/importar. - **Catálogo de Cursos/Disciplinas** (Course) com filtros por área/carga horária. - **Lista de Tarefas** (Task/Todo) com prioridade, marcação de concluído e exportação.

> É permitido propor outro domínio **equivalente** (pequeno e CRUD) — registre no README.

Requisitos mínimos 1) **Contratos** claros e pequenos: p.ex. `IReadRepository<T,TId>` / `IWriteRepository<T,TId>`; serviços dependem **apenas** do que usam (ISP).
2) **Composição centralizada**: catálogo/fábrica/política fora do cliente (sem `switch` espalhado).
3) **Repository**: InMemory **e** uma persistência real **(CSV ou JSON)** — escolha uma.
4) **Fluxos**: adicionar, listar, buscar por critério, atualizar, remover (com mensagens/retornos claros).
5) **Testes**: unitários com dublês (sem I/O) + integração usando arquivo temporário para a persistência real.
6) **Assíncrono (opcional, recomendado)**: métodos `async` e/ou `IAsyncEnumerable<T>` em um ponto do fluxo **com testes** de sucesso/erro/cancelamento.
7) **CLI simples ou console demo**: um `Program` que demonstre 3–5 casos de uso encadeados.
8) **README (raiz) vivo**: composição da equipe (nomes e RAs), estrutura do repo, decisões de design (bullets), como executar os cenários e os testes, e **rubrica preenchida** com autoavaliação.

Entregáveis (repo da equipe) - Pasta: `src/fase-11-mini-projeto/` com código + scripts/ artefatos.
- `docs/` com diagrama/resumo de arquitetura (1 página).
- **Testes** (unit + integração).
- **README (raiz)** atualizado (composição da equipe, sumário, execução, decisões, autoavaliação).

Snippets de referência (esqueleto mínimo)

```
public interface IReadRepository<T, TId>
{
    T? GetById(TId id);
    IReadOnlyList<T> ListAll();
}
public interface IWriteRepository<T, TId>
{
    T Add(T entity);
    bool Update(T entity);
```

```
    bool Remove(TId id);
}

public sealed record Book(int Id, string Title, string Author);

public sealed class CatalogService
{
    private readonly IReadRepository<Book,int> _read;
    private readonly IWriteRepository<Book,int> _write;
    public CatalogService(IReadRepository<Book,int> read,
IWriteRepository<Book,int> write)
        => (_read,_write) = (read,write);

    public Book Register(Book b) => _write.Add(b);
    public IReadOnlyList<Book> All() => _read.ListAll();
    public Book? FindById(int id) => _read.GetById(id);
    public bool Rename(int id, string newTitle)
    {
        var b = _read.GetById(id);
        if (b is null) return false;
        return _write.Update(b with { Title = newTitle });
    }
}
```

Critérios de avaliação (rubrica, 0–24) - **Contratos e composição** (cliente depende só do contrato; política centralizada) — 0–6
- **Repository** (InMemory + CSV/JSON) correto e bem isolado — 0–6
- **Testes** (unit com dublês + integração com temp file; cobertura de casos críticos) — 0–6
- **Documentação e demo** (README completo, equipe, decisão, execução; CLI funcional) — 0–6

Tempo sugerido: 1 aula + extra (ex.: 50–90 min em sala para desenho/pares, restante como entrega).
Pitfalls: misturar regra de negócio no Repository; não centralizar composição; testes com I/O em unit; interface gorda sem ISP; ausência de demo mínima.

Peso sugerido: 24/100 — consolidação prática e integrada do módulo (ajuste final de pesos conforme somatório do curso).