Fase 8 — ISP (Interface Segregation Principle)

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/`.
- Arquivo `.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 (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.