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 cubram **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/`.
- Arquivo `.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/contador).
- **README (raiz)** atualizado: **composição da equipe (nomes e RAs)**, sumário com link, como executar os testes.

Snippets C# (modelo mínimo)

Contratos mínimos

```
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 de bombeamento (leitura → escrita) com retentativa

```
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 em teste
*/ }
            }
        }
        return count;
    }
}
```

Dublês (ideias rápidas)

- **Reader fake** emitindo 3 itens via `IAsyncEnumerable<T>`.
- **Writer stub** que falha 2x e depois escreve (para acionar retentativa sem `Sleep`).
- **Clock fake** com `Now` controlado para validar backoff/política de tempo.

Testes (roteiro sugerido)

1) **Sucesso simples**: reader → 3 itens; writer sempre escreve → `RunAsync` retorna 3. 2) **Retentativa**: writer falha 2x e depois escreve → `RunAsync` retorna 3, sem `Thread.Sleep` real. 3) **Cancelamento**: cancelar após 1 item → espera-se `OperationCanceledException` e contagem parcial. 4) **Stream vazio**: reader emite 0 itens → retorno 0 sem erro. 5) **Erro no meio do stream**: reader lança exceção no 2º item → verificar propagação/registro.

Critérios de avaliação (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 (lento/não determinístico) — prefira **clock fake** ou **contador**.
- Misturar I/O real (arquivos/rede) — usar **dublês em memória**.
- Ignorar **cancelamento** e **erro no meio de stream**.

Peso sugerido

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