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/design** 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/`.
- Arquivo `.md` com **5 a 7 cheiros** (mínimo 5) contendo, para cada um: 1) **Descrição do cheiro** no contexto do projeto; 2) **Antes (snippet curto)**; 3) **Depois (snippet curto)**; 4) **Antídoto aplicado** e **princípio** associado (ex.: ISP, DIP, SRP, OCP); 5) **Teste** que demonstra o efeito/segurança (mantendo comportamento).
- **README (raiz)** atualizado: **composição da equipe (nomes e RAs)**, sumário com link, como executar testes e uma seção "decisões de design" resumindo refatorações.

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 (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” (evite; foque em **pequenas mudanças seguras**).
- Trocar um cheiro por outro (ex.: remover downcast e criar interface gorda).
- Não acompanhar com **teste** — cada antídoto precisa de prova automatizada.

Peso sugerido

10/100 — consolida qualidade contínua com melhorias pequenas, seguras e guiadas por princípios.