Trabalho Prático — OS-Lite (Classes/Records/ Structs e Enums, com Navegabilidade Bidirecional)

> **Restrições**: não usar **herança** nem **polimorfismo**; não usar **associações N:M**. Utilizar somente **1:1** e **1:N**. Trabalhar **TDD** e **invariantes**.

---

1) Contexto do problema

Uma pequena assistência técnica deseja registrar **ordens de serviço** para consertos de equipamentos. Cada **OrdemDeServico** pertence a **um Cliente** (1:N), e cada ordem contém **vários ItensDeServico** (1:N por **composição**). O objetivo é construir um **modelo de domínio enxuto, expressivo e seguro**, com **tipos adequados**, **invariantes explícitos** e **testes**.

Fora do escopo: pagamentos, estoque, catálogos complexos, autenticação, interface gráfica. O foco é **domínio + testes**.

---

2) Objetivos de aprendizagem

- Escolher e justificar o uso de **class × record/record struct × enum** com base na semântica de **identidade** vs. **valor**.
- Modelar **associações 1:N** (incluindo **composição**) e **um caso com navegabilidade bidirecional** (sem N:M).
- Proteger **invariantes** com **fail-fast** por meio de exceções adequadas e cobrir com **TDD** (casos felizes e de falha).
- **Derivar** informações (p.ex., **Total** da OS) em vez de persistir campos redundantes.

---

3) Escopo funcional (MVP)

Entidades e VOs (sem herança, sem polimorfismo)

- **Cliente** (entidade): `Id (int)`, `Nome`, `Email`, `Telefone`, **coleção de Ordens** (para navegabilidade bidirecional).
- **OrdemDeServico** (entidade): `Id (int)`, `ClienteId`, `Cliente` (navegabilidade de volta), `DataAbertura (DateOnly)`, `Status (enum)`, **coleção de ItensDeServico** (interna/ encapsulada), `Total` **derivado**.
- **ItemDeServico** (parte por **composição** da Ordem): `Descricao`, `Quantidade (int>0)`, `PrecoUnitario (Money)`, *(opcional)* `Categoria (enum)`.
- **Money** (VO): valor monetário **não negativo**; preferir **record struct** e imutabilidade.
- **Email** (VO opcional): validação simples de formato.

Enums

- `StatusOS { Aberta, EmExecucao, Concluida, Cancelada }`
- `CategoriaItem { Diagnostico, Pecas, MaoDeObra }` *(opcional)*

Casos de uso mínimos

1. **Abrir OS** para um cliente existente (status inicial `Aberta`).
2. **Adicionar/Remover item** na OS (permitido apenas quando `Status` ∈ {`Aberta`, `EmExecucao`}).
3. **Iniciar execução** (`Aberta` → `EmExecucao`) — exige **≥ 1 item**.
4. **Concluir** (`EmExecucao` → `Concluida`).
5. **Cancelar** (`Aberta` / `EmExecucao` → `Cancelada`).
6. **Obter Total** da OS = soma de `PrecoUnitario * Quantidade` (campo **derivado**, sem setter).

---

4) Associações e navegabilidade bidirecional

4.1 Cliente ↔ OrdemDeServico (1:N, bidirecional)

- **No Cliente**: manter `IReadOnlyCollection<OrdemDeServico>` exposta; **mutações via métodos controlados** (p.ex., `AdicionarOrdem`, `RemoverOrdem`).
- **Na Ordem**: manter `ClienteId` e referência `Cliente`.
- **Consistência**: ao **vincular** uma OS a um Cliente, **sincronizar ambos os lados** (definir o `Cliente` / `ClienteId` e atualizar a coleção do cliente). Ao **trocar** de cliente, **remover** da coleção do antigo e **adicionar** na coleção do novo — sem estados intermediários inválidos.
- **Criação coesa**: fornecer um ponto de criação que devolva a OS já ligada e com `Status = Aberta`.
- **Encapsulamento**: não expor setters públicos nas coleções; evitar que código externo quebre a consistência.

4.2 OrdemDeServico → ItensDeServico (1:N, composição)

- Itens **nascem e morrem** com a OS.
- Coleção interna controlada por operações (`AdicionarItem`, `RemoverItem`).
- **Total** é **derivado** da coleção atual.

> **Importante**: a navegabilidade bidirecional **não cria N:M**; permanece **1:N** (um Cliente tem muitas Ordens; cada Ordem tem exatamente um Cliente).

---

5) Invariantes e validações (fail-fast)

- **Money**: `Value >= 0`. Caso contrário, lançar exceção adequada (p.ex., `ArgumentOutOfRangeException`).
- **ItemDeServico**: `Quantidade > 0`, `Descricao` não vazia; subtotal calculado a partir de VO; categoria (quando usada) deve ser enum válido.
- **OrdemDeServico**:
- **Transições**:
  - `Aberta` → `EmExecucao`: exige **≥ 1 item**.

    - `EmExecucao` → `Concluida`: permitido.
    - `Aberta|EmExecucao` → `Cancelada`: permitido.
- **Proibições**: não adicionar/remover itens em `Concluida` ou `Cancelada`.
- **Total derivado**: sem setter e sem persistência redundante.
- **Cliente**: `Nome` não vazio; `Email` válido (se VO for adotado).
- **Bidirecional**: vínculos **sempre consistentes** (sem divergência entre `ClienteId`, referência `Cliente` e coleções).

---

6) Critérios de aceite

- **Fluxo de status** cumpre todas as regras e bloqueios.
- **Total** da OS correto para combinações de itens (incluindo casos de borda).
- **Exceções específicas** e mensagens claras ao violar invariantes (fail-fast).
- **Sem herança/polimorfismo**; **sem N:M**.
- **Tipos adequados** (entidades como classes; VOs como records/record struct; enums para estados/tipos).
- **Encapsulamento**: coleções imutáveis externamente; mutações via métodos da entidade/agregado.
- **Navegabilidade bidirecional** funcional e **consistente** (cobrir criação, troca de cliente e remoção).

---

7) Roteiro TDD (ordem sugerida de testes)

> Use a tríade **vermelho → verde → refatorar**; nomeie testes de forma descritiva.

1. `Money_nao_aceita_negativo`.
2. `ItemDeServico_cria_valido_e_calcula_subtotal`.
3. `OS_total_soma_subtotais_itens`.
4. `OS_aberta_inicia_execucao_quando_tem_itens` e `OS_aberta_nao_inicia_sem_itens`.
5. `OS_nao_adiciona_itens_em_concluida_ou_cancelada`.
6. `OS_fluxo_aberta_para_execucao_para_concluida`.
7. **Bidirecional** — `Cliente_adiciona_ordem_sincroniza_cliente_na_ordem`.
8. **Bidirecional** — `OS_trocar_de_cliente_atualiza_colecoes_dos_clientes` (remove do antigo, adiciona no novo e atualiza `ClienteId`).
9. `Email_invalido_deve_falhar` *(se VO for adotado)*.

---

8) Entregáveis

1. **Solução .NET** com, no mínimo, dois projetos: `OSLite.Domain` e `OSLite.Domain.Tests` (console opcional para demonstração).
2. **Código limpo** com **invariantes protegidos** e **testes passando** (incluindo casos de falha e borda).
3. **README** justificando as decisões de modelagem: onde usou **class × record/record struct × enum**, e por quê.
4. **Reflexão** escrita (seção 10).

9) Rubrica de avaliação (10 pts)

- **Modelagem adequada de tipos** (class × record/record struct × enum): **2,0**
- **Invariantes e exceções claras (fail-fast)**: **2,0**
- **Associações e fluxo de status**: **2,0**
- **Navegabilidade bidirecional consistente + testes**: **2,0**
- **Testes (felizes + falhas; nomes claros)**: **1,5**
- **Reflexão crítica (clareza, pertinência, exemplos)**: **0,5**

Penalizações: exposição indevida de setters, coleções mutáveis externamente, `Total` armazenado em vez de derivado, ausência de testes de falha.

---

10) Reflexão final (entrega textual — 10 a 15 linhas)

Tema: *Como records/structs e enums impactam o processo de desenvolvimento e por que importam?*

Guia (utilize exemplos do seu próprio código): - Diferencie **identidade** (entidades → classes) de **semântica de valor** (VOs → record/record struct) e discuta impactos em **igualdade**, **imutabilidade**, **refatorações** e **testes**. - Explique como **enums** tornam **estado e regras** explícitos (evitando "strings mágicas") e como isso melhora **legibilidade** e **validação**. - Relate como **invariantes + fail-fast** facilitaram o TDD (testes de falha objetivos) e reduziram ambiguidade. - Reflita sobre benefícios e cuidados da **navegabilidade bidirecional** (sincronismo dos dois lados, encapsulamento, prevenção de estados inválidos).

---

11) Dicas práticas

- **Coleções**: exponha `IReadOnlyCollection<T>`; modifique via métodos da entidade.
- **Bidirecional**: concentre o vínculo num único ponto para garantir consistência.
- **Mensagens de erro**: curtas e específicas (ajudam a entender a regra violada).
- **Testes de falha**: nomeie explicitamente a regra/invariante que deve ser protegida.
- **Refatoração**: após verde, considere extrair *guard clauses* e validar construtores enxutos.

---

12) Extensões opcionais (apenas após o MVP)

- VO **Telefone** (normalização/chaves permitidas).
- VO **DescricaoDeItem** (mín./máx. de caracteres).
- Relatório textual simples listando ordens por cliente com seus totais (via console ou testes).

---

13) Entrega

- Enviar repositório/pasta com `OSLite.Domain/`, `OSLite.Domain.Tests/`, README e a **reflexão**.
- Garantir que os **testes rodam** em máquina limpa (instruções no README).

Relembrando: sem herança/polimorfismo; sem N:M; foco em classes/records/structs, enums, associações 1:N (composição), navegabilidade bidirecional (1:N), invariantes e TDD.