Lousa — Fase 5: Repository InMemory (contrato + implementação em coleção)

1. Visão geral da fase

- **Nome da fase:** Fase 5 — Repository (InMemory) — contrato único para acesso a dados
- **Meta central:** Introduzir o **padrão Repository** como **ponto único de acesso a dados** da equipe, usando um domínio simples (ex.: `Book`, `Student`, `Order`) com implementação **InMemory** baseada em coleção.
- **Ideia-chave:** O **cliente fala só com o Repository**, nunca com coleções diretamente.

---

2. Título e descrição (para ClassHero)

- **Título (ClassHero):** Fase 5 — Repository InMemory (contrato + implementação em coleção)
- **Descrição (ClassHero):**

> Introduzam o padrão Repository como ponto único de acesso a dados da equipe (domínio simples, por exemplo: `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).

---

3. Entregáveis (no repositório único da equipe)

- **Pasta da fase:** `src/fase-04-repository-inmemory/`
- **Arquivo `.md` da fase:**
- Diagrama leve (pode ser ASCII ou imagem simples) com **Cliente → Repository → Coleção InMemory**.
- Snippets C# mostrando: **contrato**, **implementação InMemory** e **uso em um cliente/serviço**.
- **Testes unitários:**
- Cobrir operações básicas e casos de erro (ex.: `id` não encontrado).
- `README.md` **(na raiz) atualizado:**
- Composição da equipe (nomes completos e RAs).
- Sumário com link/âncora para a Fase 5.
- Instruções de como executar os **testes** da Fase 5.

---

4. Contrato do Repository (modelo genérico)

```
public interface IRepository<T, TId>
{
    T Add(T entity);
    T? GetById(TId id);
```

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

Pontos importantes: - Expõe só **operações de acesso a dados**, sem regra de negócio. - Retorna `IReadOnlyList<T>` para não expor coleções mutáveis.

---

5. Implementação InMemory (coleção + política de id)

Exemplo simples usando `Dictionary<TId, T>` e um `Func<T, TId>` para extrair o identificador.

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

    public InMemoryRepository(Func<T, TId> getId)
    {
        _getId = getId ?? throw new ArgumentNullException(nameof(getId));
    }

    public T Add(T entity)
    {
        var id = _getId(entity);
        _store[id] = entity;
        return entity;
    }

    public T? GetById(TId id)
    {
        return _store.TryGetValue(id, out var entity) ? entity : default;
    }

    public IReadOnlyList<T> ListAll()
    {
        return _store.Values.ToList();
    }

    public bool Update(T entity)
    {
        var id = _getId(entity);
        if (!_store.ContainsKey(id))
            return false;

        _store[id] = entity;
        return true;
```

```
    }

    public bool Remove(TId id)
    {
        return _store.Remove(id);
    }
}
```

Observações didáticas: - Esta implementação é **somente em memória** (sem arquivo, sem BD, sem I/O). - A **política de** `id` (quem gera o id) deve ser **decidida e documentada** pela equipe: - Ou o id vem de fora (ex.: já existe no domínio). - Ou o Repository gera o próximo id (incremental, GUID etc.).

---

6. Domínio de exemplo (Book) e uso pelo cliente

6.1. Modelo de domínio

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

6.2. Serviço de domínio que fala só com o Repository

```
public static class BookService
{
    public static Book Register(IRepository<Book, int> repo, Book book)
    {
        // Aqui poderia haver validações de domínio (título obrigatório,
etc.)
        return repo.Add(book);
    }

    public static IReadOnlyList<Book> ListAll(IRepository<Book, int> repo)
    {
        return repo.ListAll();
    }
}
```

6.3. Composição em um ponto único (exemplo Program)

```
public static class Program
{
    public static void Main(string[] args)
    {
        // Fábrica simples do Repository para Book
        IRepository<Book, int> repo =
            new InMemoryRepository<Book, int>(book => book.Id);

        // Uso através do serviço, sem tocar na coleção diretamente
```

```
        BookService.Register(repo, new Book(1, "Código Limpo", "Robert C.
Martin"));
        BookService.Register(repo, new Book(2, "Domain-Driven Design", "Eric
Evans"));

        var all = BookService.ListAll(repo);

        Console.WriteLine("Livros cadastrados:");
        foreach (var book in all)
        {
            Console.WriteLine($"#{book.Id} - {book.Title} ({book.Author})");
        }
    }
}
```

---

7. Testes unitários (exemplo em xUnit)

Critérios sugeridos no enunciado: - `Add/ListAll` retorna 1 item após inserção. - `GetById` para id existente retorna entidade; para id ausente, `null`. - `Update` devolve `true` quando existe; `false` caso contrário. - `Remove` devolve `true` quando remove; `false` caso contrário. - Cliente **não acessa coleções diretamente**, apenas via `Repository`.

7.1. Snippets de teste

```
public class InMemoryRepositoryTests
{
    private static InMemoryRepository<Book, int> CreateRepo()
        => new InMemoryRepository<Book, int>(b => b.Id);

    [Fact]
    public void Add_Then_ListAll_ShouldReturnOneItem()
    {
        var repo = CreateRepo();

        repo.Add(new Book(1, "Livro A", "Autor"));
        var all = repo.ListAll();

        Assert.Single(all);
        Assert.Equal(1, all[0].Id);
    }

    [Fact]
    public void GetById_Existing_ShouldReturnEntity()
    {
        var repo = CreateRepo();
        repo.Add(new Book(1, "Livro A", "Autor"));

        var found = repo.GetById(1);
```

```
        Assert.NotNull(found);
        Assert.Equal("Livro A", found!.Title);
    }

    [Fact]
    public void GetById_Missing_ShouldReturnNull()
    {
        var repo = CreateRepo();

        var found = repo.GetById(99);

        Assert.Null(found);
    }

    [Fact]
    public void Update_Existing_ShouldReturnTrue()
    {
        var repo = CreateRepo();
        repo.Add(new Book(1, "Livro A", "Autor"));

        var updated = repo.Update(new Book(1, "Livro A (Revisto)", "Autor"));

        Assert.True(updated);
        Assert.Equal("Livro A (Revisto)", repo.GetById(1)!.Title);
    }

    [Fact]
    public void Update_Missing_ShouldReturnFalse()
    {
        var repo = CreateRepo();

        var updated = repo.Update(new Book(1, "Livro A", "Autor"));

        Assert.False(updated);
    }

    [Fact]
    public void Remove_Existing_ShouldReturnTrue()
    {
        var repo = CreateRepo();
        repo.Add(new Book(1, "Livro A", "Autor"));

        var removed = repo.Remove(1);

        Assert.True(removed);
        Assert.Empty(repo.ListAll());
    }

    [Fact]
    public void Remove_Missing_ShouldReturnFalse()
```

```
    {
        var repo = CreateRepo();

        var removed = repo.Remove(99);

        Assert.False(removed);
    }
}
```

---

8. 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**

---

9. Tempo e peso sugeridos

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

---

10. Pitfalls a evitar (para discutir com a turma)

- Colocar **regra de negócio** dentro do Repository (mantenha-o focado em acesso a dados).
- Expor **coleções mutáveis** (use `IReadOnlyList<T>`).
- Misturar política de `id` sem documentar (é o domínio que fornece ou o Repository que gera?).