Lousa — Fase 4: Interface plugável e testável (contrato + composição + dublês)

1. Visão geral da fase

- **Nome da fase:** Fase 4 — Interface plugável e testável
- **Meta central:** Evoluir a solução da Fase 3 introduzindo:
- Um **contrato explícito** (interface) para o passo variável.
- Um **ponto único de composição** (política → implementação concreta).
- **Cliente** dependendo apenas do contrato (não conhece as classes concretas).
- **Teste com dublês** (fake/stub) sem I/O.

---

2. Enunciado (para ClassHero)

> Evoluam a Fase 3 introduzindo um contrato explícito para o passo variável e um ponto único de composição (política → implementação). O cliente deve depender do contrato, não dos concretos. Demonstrem como testar o cliente usando dublês (fake/stub) sem I/O.

---

3. Entregáveis no repositório da equipe

Pasta da fase - `src/fase-03-com-interfaces/`

Artefato principal (.md) - Um arquivo `.md` com trechos de código C# mostrando, obrigatoriamente: 1. **Contrato** (interface) para o passo variável. 2. **Duas (ou mais) implementações** concretas desse contrato. 3. **Cliente** dependendo apenas do contrato (recebe a interface, não a concreta). 4. **Composição** em um ponto único (catálogo/fábrica simples) que converte a política/mode em implementação concreta. 5. **Teste com dublê** (fake/stub) mostrando como testar o cliente sem I/O real.

README na raiz do repositório (atualizado) - (a) Composição da equipe (nomes completos e RAs). - (b) Sumário com link/âncora para a Fase 4. - (c) Instruções sobre **como executar** os exemplos de uso e os **testes** da Fase 4.

---

4. Estrutura conceitual da solução

1. **Contrato (interface)**
2. Descreve o **"o que"** a peça faz (comportamento esperado), não o "como".
3. Ex.: `ITextFormatter`, `INotificationSender`, `IFreightCalculator`.
4. **Implementações concretas**
5. Cada classe concreta implementa o contrato com um comportamento específico.

6. Ex.: `UpperCaseFormatter`, `LowerCaseFormatter`, `TitleCaseFormatter`, etc.

7. **Cliente dependente do contrato**

8. O cliente recebe a interface via construtor/método (injeção de dependência).

9. Não deve instanciar concretos diretamente.

10. **Ponto de composição (catálogo/fábrica)**

11. Local único que decide qual implementação concreta usar com base em uma política (ex.: `mode` vindo de string).

12. Ex.: `FormatterCatalog.Resolve(string mode)` que devolve `ITextFormatter`.

13. **Teste com dublê**

14. Criar fake/stub implementando a mesma interface para testar apenas a lógica do cliente.
15. Sem acessar I/O, sem criar objetos pesados.

---

5. Sugestão de organização dos snippets

- **Bloco 1 — Interface:** assinatura enxuta, foco no comportamento variável.
- **Bloco 2 — Concretas:** 2 a 4 classes seladas (`sealed`) implementando o contrato.
- **Bloco 3 — Cliente:** classe/método que recebe o contrato por parâmetro/ construtor.
- **Bloco 4 — Composição:** catálogo/factory que converte `mode` → implementação.
- **Bloco 5 — Teste com dublê:** teste unitário que injeta um fake e verifica o comportamento do cliente.

---

6. Critérios implícitos da fase (ligação com o módulo)

- Cliente **não conhece** classes concretas → depende apenas de interface.
- **Composição centralizada** (sem `new` espalhado, sem `switch` no cliente).
- Uso de **dublês** em testes para evitar I/O e tornar o teste rápido e determinístico.
- Fase vista como **marco da alternância verdadeira**: contrato + composição + testes com dublês.

---

7. Peso sugerido na avaliação

- **Peso sugerido:** 10/100
- Papel da fase na jornada: consolidar a ideia de **interfaces plugáveis**, preparando o terreno para o **Repository** (InMemory, CSV, JSON) das fases seguintes.

8. Exemplo completo em C# (contrato + concretas + cliente + composição + dublê)

```
// 1) Contrato para o passo variável
public interface ITextFormatter
{
    string Format(string input);
}

// 2) Implementações concretas do contrato
public sealed class UpperCaseFormatter : ITextFormatter
{
    public string Format(string input) => input.ToUpperInvariant();
}

public sealed class LowerCaseFormatter : ITextFormatter
{
    public string Format(string input) => input.ToLowerInvariant();
}

public sealed class TitleCaseFormatter : ITextFormatter
{
    public string Format(string input)
    {
        if (string.IsNullOrWhiteSpace(input)) return input;

        var words = input.Split(' ');
        for (int i = 0; i < words.Length; i++)
        {
            if (words[i].Length == 0) continue;
            var first = char.ToUpper(words[i][0]);
            var rest = words[i].Substring(1).ToLowerInvariant();
            words[i] = first + rest;
        }
        return string.Join(" ", words);
    }
}

// 3) Cliente que depende apenas da interface (injeção de dependência)
public sealed class MessageService
{
    private readonly ITextFormatter _formatter;

    public MessageService(ITextFormatter formatter)
    {
        _formatter = formatter ?? throw new
ArgumentNullException(nameof(formatter));
    }

    public string BuildMessage(string name)
```

```
    {
        var raw = $"Olá, {name}! Bem-vindo ao sistema.";
        return _formatter.Format(raw);
    }
}

// 4) Ponto único de composição (catálogo / factory simples)
public static class FormatterCatalog
{
    public static ITextFormatter Resolve(string mode)
    {
        return mode?.ToLowerInvariant() switch
        {
            "upper" => new UpperCaseFormatter(),
            "lower" => new LowerCaseFormatter(),
            "title" => new TitleCaseFormatter(),
            _ => new UpperCaseFormatter(), // padrão seguro
        };
    }
}

// 5) Exemplo de uso no Program.cs
public class Program
{
    public static void Main(string[] args)
    {
        Console.Write("Informe o modo (upper, lower, title): ");
        var mode = Console.ReadLine();

        Console.Write("Informe seu nome: ");
        var name = Console.ReadLine();

        // composição em um único ponto
        ITextFormatter formatter = FormatterCatalog.Resolve(mode);
        var service = new MessageService(formatter);

        var message = service.BuildMessage(name ?? string.Empty);
        Console.WriteLine();
        Console.WriteLine("Mensagem gerada:");
        Console.WriteLine(message);
    }
}

// 6) Teste com dublê (fake) para o cliente
// Exemplo em xUnit, mas pode ser adaptado para qualquer framework
public sealed class FakeFormatter : ITextFormatter
{
    public string LastReceivedInput { get; private set; } = string.Empty;

    public string Format(string input)
    {
```

```
        LastReceivedInput = input;
        // devolve algo previsível para o teste
        return $"[FAKE]{input}[FAKE]";
    }
}

public class MessageServiceTests
{
    [Fact]
    public void BuildMessage_DeveUsarFormatterInjetado()
    {
        // arrange
        var fake = new FakeFormatter();
        var service = new MessageService(fake);

        // act
        var result = service.BuildMessage("Everton");

        // assert (exemplos de verificações)
        Assert.Equal("Olá, Everton! Bem-vindo ao sistema.",
fake.LastReceivedInput);
        Assert.Equal("[FAKE]Olá, Everton! Bem-vindo ao sistema.[FAKE]",
result);
    }
}
```