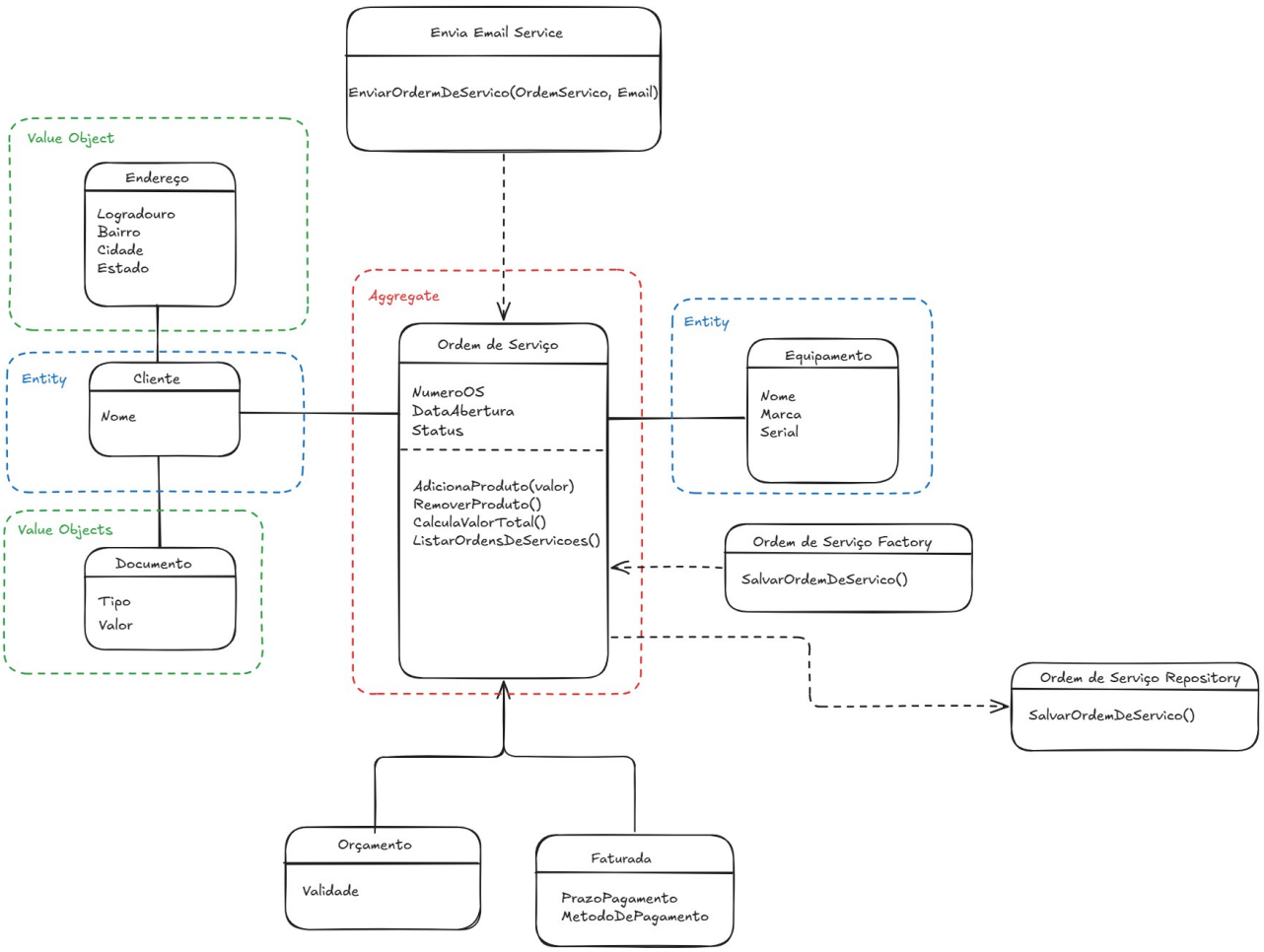
Tecnologia .NET [25E2_2]

Contexto: Em uma assistência técnica, diariamente dezenas de equipamentos dão entrada para se realizar análises técnicas a fim de detectar problemas em seus componentes. Mediante a essa análise técnica é possível estimar/propor um orçamento para o cliente ou ate sugerir a compra de um novo equipamento. Mediante a esse orçamento o cliente pode optar por realizar a manutenção ou adquirir um novo equipamento.

Projeto Para melhorar e agilizar o processo de orçamento foi proposto a criação de um sistema que receba como entrada os dados do cliente e valor do(s) equipamento(s), e gere um documento detalhando a composição da ordem de serviço e o envie por email para o cliente.

Parte 1

Modelagem



Código

https://github.com/Diego-MP/Tecnologia-.NET-25E2_2-

Parte 2

- Implementação das classes considerando os conceitos básicos de OO como Encapsulamento, Abstração, Herança e Polimorfismo;

Encapsulamento

Métodos de aceso dos atributos Cliente, Equipamento e OrdemDeServico

Abstração

Exemplo da interface

```
namespace Clipper.OS._3.DomainLayer.Repositorie;

3 usages  1 inheritor  Diego M.P
public interface IOrdemDeServicoRepositoy
{
    1 implementation  Diego M.P
    public void AdicionarOrdemDeServico(OrdemDeServico ordemDeServico);

    1 usage  1 implementation  Diego M.P
    public OrdemDeServico  BuscarOrdemDeServico(int numeoDaOrdemDeServico);

    1 implementation  Diego M.P
    public List<OrdemDeServico> ListarTodas();
}
```

Herança

A implementação dos métodos da Interface

```
namespace Clipper.OS._4.InfrastructureLayer;

6 usages  Diego M.P
public class OrdemDeServicoRepositoy : IOrdemDeServicoRepositoy
{
    private static readonly List<OrdemDeServico> _ordens = new();

    7 usages  Diego M.P
    public void AdicionarOrdemDeServico(OrdemDeServico ordemDeServico)
    {
        if (ordemDeServico == null)
            throw new ArgumentNullException(nameof(ordemDeServico));
        _ordens.Add(ordemDeServico);
    }

    1+1 usages  Diego M.P
    public OrdemDeServico  BuscarOrdemDeServico(int id)
    {
        return _ordens.FirstOrDefault(o :OrdemDeServico  => o.NumeroOS == id);
    }
```

Polimorfismo

Sobrecarga dos métodos herdados da Interface

```
public class Orcamento : OrdemDeServico
{
    1 usage
    public DateTime Validade { get; set; }

    1 usage  Diego M.P
    public Orcamento()
    {

    }

    1+2 usages  Diego M.P
    public override void EnviarOrdemDeServico(OrdemDeServico ordemDeServico, string
    {
        Console.WriteLine("[Email]: Enviando Orçamento da ordem de serviço...");

    }
}
```

- Implementação as classes e objetos em C#, utilizando modificadores de acesso, propriedades, métodos e construtores;

- Utilização de herança e polimorfismo em C# para criar hierarquias de classes flexíveis e extensíveis;

- Aplicação de abstração e encapsulamento em C# para ocultar detalhes de implementação e expor interfaces claras e concisas.

Parte 3

- Aplicação dos princípios SOLID no design de classes, criando classes coesas, com alta responsabilidade e baixo acoplamento;
- Utilização do princípio de Single Responsibility nas implementações das Classes;
- Utilização do padrão Low Coupling nas classes para garantir o baixo acoplamento;
- Utilização do padrão High Cohesion nas classes para garantir que elas apresentam coesão.

Para cada princípio e padrão utilizado deve-se:

- Destacar o trecho em que o princípio e/ou padrão foi aplicado, podendo pintá-lo, colocá-lo em negrito ou sublinhá-lo;
- Informar o nome e o objetivo do princípio;
- Explicar o trecho do código em que o aplica, através de comentários.

Princípio da Responsabilidade Única

A classe EnviarOrdemDeServico tem uma única responsabilidade: orquestrar o envio de uma ordem de serviço por e-mail, delegando persistência e envio para outras classes.

```
/*Princípio SOLID: Single Responsibility Principle (SRP) — Princípio da Responsabilidade Única
A classe EnviarOrdemDeServico tem uma única responsabilidade: orquestrar o envio de uma ordem
de serviço por e-mail, delegando persistência e envio para outras classes.
*/
1 usage  Diego M.P
public class EnviarOrdemDeServico
{
    private IOrdemDeServicoRepositoy _ordemDeServicoRepositoy;
    private EnviarOrdemDeServicoService _enviarOrdemDeServicoService;

    1 usage  Diego M.P
    public EnviarOrdemDeServico(IOrdemDeServicoRepositoy ordemDeServicoRepositoy,
        EnviarOrdemDeServicoService enviarOrdemDeServicoService)
    {
        _ordemDeServicoRepositoy = ordemDeServicoRepositoy;
        _enviarOrdemDeServicoService = enviarOrdemDeServicoService;
    }

    1 usage  Diego M.P
    public bool Enviar(int ordemDeServicoId, string email)
    {
        try
        {
            OrdemDeServico ordemDeServico = _ordemDeServicoRepositoy.BuscarOrdemDeServico(ordemDeServicoId);
            _enviarOrdemDeServicoService.Enviar(ordemDeServico, email);
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Erro ao enviar ordem de serviço: {ex.Message}");
            return false;
        }
    }
}
```

Alta Coesão

A classe OrdemDeServicoRepositoy é altamente coesa, pois todos os seus métodos tratam exclusivamente de operações sobre ordens de serviço.

```
namespace Clipper.OS._4.InfrastructureLayer;
/*
 * Alta Coesão
 * A classe OrdemDeServicoRepositoy é altamente coesa, pois todos os seus métodos tratam
 * exclusivamente de operações sobre ordens de serviço.
 */
6 usages  Diego M.P
public class OrdemDeServicoRepositoy : IOrdemDeServicoRepositoy
{
    private static readonly List<OrdemDeServico> _ordens = new();

    7 usages  Diego M.P
    public void AdicionarOrdemDeServico(OrdemDeServico ordemDeServico)
    {
        if (ordemDeServico == null)
            throw new ArgumentNullException(nameof(ordemDeServico));
        _ordens.Add(ordemDeServico);
    }

    1+1 usages  Diego M.P
    public OrdemDeServico  BuscarOrdemDeServico(int id)
    {
        return _ordens.FirstOrDefault(o :OrdemDeServico  => o.NumeroOS == id);
    }

    1 usage  Diego M.P
    public List<OrdemDeServico> ListarTodas()
    {
        return _ordens.ToList();
    }
}
```

Baixo Acoplamento

A utilização de interfaces e injeção de dependência reduz o acoplamento entre as classes, permitindo trocar implementações facilmente.

```
public class EnviarOrdemDeServico
{
    private IOrdemDeServicoRepositoy _ordemDeServicoRepositoy;
    private EnviarOrdemDeServicoService _enviarOrdemDeServicoService;

    1 usage  Diego M.P *
    public EnviarOrdemDeServico(IOrdemDeServicoRepositoy ordemDeServicoRepositoy,
        EnviarOrdemDeServicoService enviarOrdemDeServicoService)
    {
        /* Baixo Acoplamento
         * As dependências são injetadas, não criadas internamente.
         * A utilização de interfaces e injeção de dependência reduz o acoplamento entre as classes,
         * permitindo trocar implementações facilmente.
         */
        _ordemDeServicoRepositoy = ordemDeServicoRepositoy;
        _enviarOrdemDeServicoService = enviarOrdemDeServicoService;
    }
```

Parte 4

Foi utilizado o xUnit devido à incompatibilidade do ambiente e do IDE com o MSTest.

As classes para teste de Cliente:

Nao_Deve_Aceitar_Documento_Sem_Tipo()

Testa se o documento inserido possui um tipo associado.

Deve_Aceitar_Documento_Valido()

Testa se os dados inseridos são validos (no caso de demonstração somente verifica se foram inseridos)

As classes para teste de Equiapemnto

Deve_Adicionar_Equipamento_Valido()

Testa se os dados inseridos são válidos (no caso de demonstração somente verifica se foram inseridos)

Nao_Deve_Aceitar_Serial_Nulo()

Testa se o valor do produto não é nulo

Deve_Aceitar_Valor_Zero()

Testa se o produto aceita o valor 0.

Nao_Deve_Aceitar_Valor_Negativo()

Testa se o valor passado é negatico (somente aceito valores >= 0)

As classes para teste de Ordem De Serviço

Deve_Criar_OrdemDeServico_Valida()

Testa se huve a criação correta da ordem de serviço com os dados necessarios

Nao_Deve_Criar_OrdemDeServico_SemCliente()

Testa se a ordem de servico possui um cliente associado

Nao_Deve_Enviar_OrdemDeServico_Orcamento_Sem_OrdemDeServico()

Testa se o envio de ordem de serviço do tipo orçamento possui uma ordem de serviço

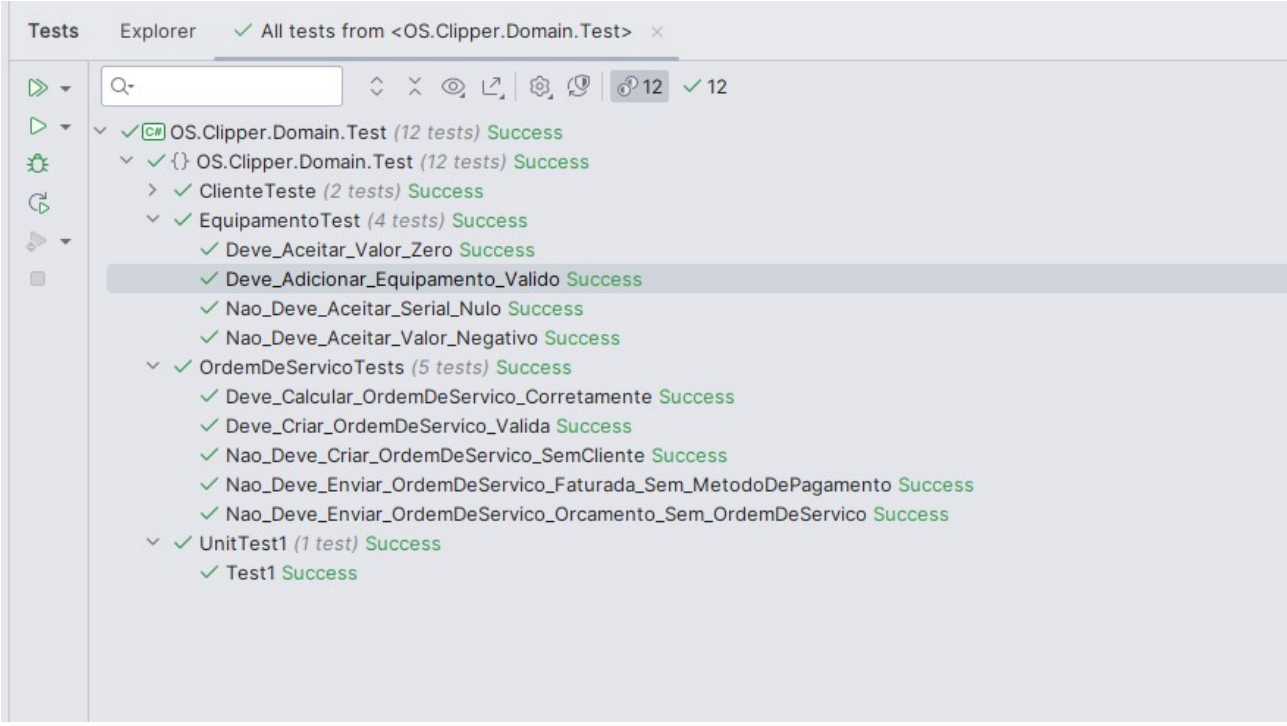
Nao_Deve_Enviar_OrdemDeServico_Faturada_Sem_MetodoDePagamento()

`Testa se o envio de ordem de serviço do tipo faturada  possui uma ordem de serviço`

Deve_Calcular_OrdemDeServico_Corretamente()

`Testa o calculo (somatória) de uma ordem de serviço esta correta`

Processo de excução do teste:



Código

https://github.com/Diego-MP/Tecnologia-.NET-25E2_2-