Infnet

Projeto de Bloco

Disciplina: Tecnologia .NET [25E2_2]

Aluno(a): Thamiris Fernandes Torres da Silva Freire

Repositório disponível em: https://github.com/thamirisftsinfnet/registroempresarial.git

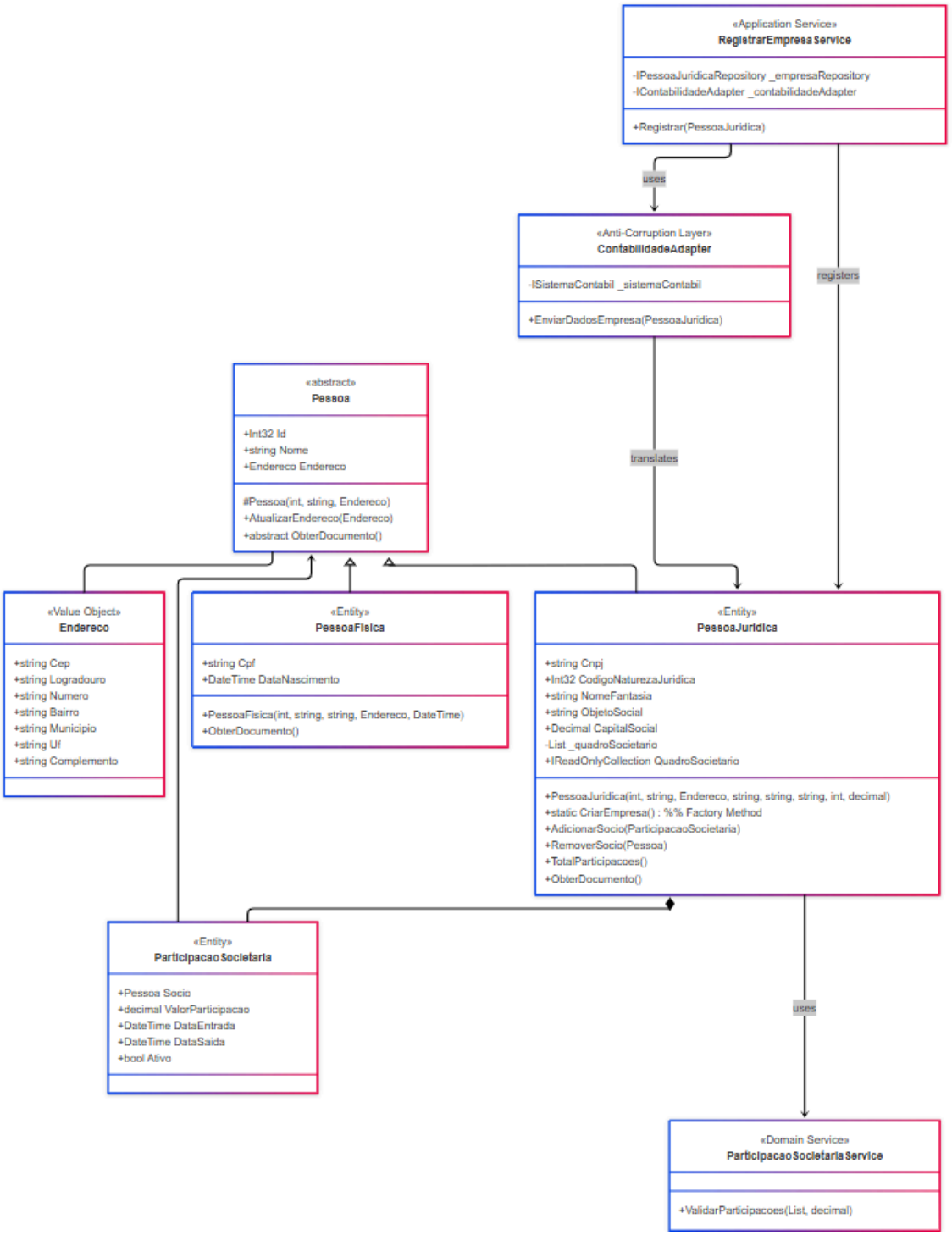
Registro de Empresas - Sistema de Cadastro para Junta Comercial

Descrição do Projeto

O projeto consiste em um sistema de cadastro de empresas para uma junta comercial, permitindo o registro completo das informações empresariais, incluindo dados cadastrais, natureza jurídica, capital social, quadro societário e objeto social.

Escopo e Problema a Resolver

O sistema visa resolver o problema de registro e atualização de informações empresariais junto à junta comercial.



https://www.mermaidchart.com/raw/3ed118a7-e893-4afe-ab93-5d03d305046f?theme=light&version=v0.1&format=svg

Modelagem do Domínio (DDD)

A modelagem do domínio foi realizada utilizando os conceitos de DDD, com foco na Ubiquitous Language e na separação clara de responsabilidades.

Ubiquitous Language

O projeto utiliza uma linguagem ubíqua clara relacionada ao domínio de registro empresarial:

- **PessoaJuridica**: Representa empresas com CNPJ, capital social, etc.
- **PessoaFisica**: Representa pessoas físicas com CPF
- **ParticipacaoSocietaria**: Representa a participação de sócios em uma empresa
- **QuadroSocietario**: Coleção de participações societárias
- **CapitalSocial**: Valor financeiro que constitui a empresa

Esta linguagem é consistente em todo o código, refletindo os termos do domínio real de registro empresarial.

Entities e Value Objects

Entities (com identidade única):

- **Pessoa** (classe abstrata base)
- **PessoaJuridica** (empresa)
- **PessoaFisica** (pessoa física)
- **ParticipacaoSocietaria** (participação de um sócio)

Todas possuem identidade única por meio de propriedades como Id, CNPJ ou CPF.

Value Objects (sem identidade, imutáveis):

- **Endereco**: Representa um endereço completo sem identidade própria

Aggregates e Aggregate Roots

- **Aggregate Root**: PessoaJuridica atua como raiz do agregado
- **Aggregate**: ParticipacaoSocietaria é um componente do agregado, acessado somente através da raiz
- O encapsulamento é mantido pela coleção privada _quadroSocietario com acesso público apenas como IReadOnlyCollection

Bounded Contexts

O código apresenta dois bounded contexts principais:

1. **RegistroEmpresarial**: Foco no registro e gerenciamento de empresas
    - Inclui PessoaJuridica, ParticipacaoSocietaria, ParticipacaoSocietariaService
2. **Contabilidade**: Foco nos aspectos contábeis
    - Representado pela interface ISistemaContabil

Domain Services

O projeto implementa serviços de domínio para operações que não pertencem naturalmente a entidades:

- ParticipacaoSocietariaService: Valida regras de negócio relacionadas às participações societárias
- RegistrarEmpresaService: Gerencia o processo de registro de empresas

Factories

O padrão Factory é implementado através do método estático CriarEmpresa na classe PessoaJuridica, que encapsula a lógica complexa de criação de uma empresa, incluindo validações e adição de sócios.

Anti-Corruption Layer (ACL)

O padrão Anti-Corruption Layer é implementado através da classe **ContabilidadeAdapter**, que atua como intermediário entre o contexto de **RegistroEmpresarial** e o sistema contábil externo, isolando os diferentes modelos de domínio.

Orientação a Objetos com C#

Encapsulamento

O encapsulamento é aplicado consistentemente em todo o código:

- Propriedades com modificadores de acesso apropriados (private, public)
- Uso de coleções privadas com acesso público restrito (_quadroSocietario e QuadroSocietario)
- Validações nos construtores para garantir a integridade dos dados

Abstração

- Pessoa é uma classe abstrata que define comportamentos comuns
- IRepository<T> define uma interface genérica para acesso a dados
- Interfaces como IParticipacaoSocietariaService definem contratos sem implementação

Herança

- PessoaFisica e PessoaJuridica herdam da classe abstrata Pessoa
- A hierarquia de classes permite compartilhar atributos e comportamentos comuns

Polimorfismo

- **Método abstrato ObterDocumento()** é implementado de forma diferente nas subclasses
- Interfaces permitem diferentes implementações dos mesmos métodos

Padrões SOLID e GRASP

Princípios SOLID

1. Single Responsibility Principle (SRP)

Objetivo: Uma classe deve ter apenas uma razão para mudar.

Aplicação: ParticipacaoSocietariaService tem a única responsabilidade de validar participações societárias:

Open/Closed Principle (OCP)

Objetivo: Entidades devem estar abertas para extensão, mas fechadas para modificação.

Aplicação: Pessoa como classe abstrata permite extensão através de novas subclasses sem modificar o código existente:

```
public abstract class Pessoa{
    // Propriedades e métodos comuns
    public abstract string ObterDocumento();
}


public class PessoaFisica : Pessoa{
    // Implementação específica
    public override string ObterDocumento() => Cpf;
}


public class PessoaJuridica : Pessoa{
    // Implementação específica
    public override string ObterDocumento() => Cnpj;
}
```

3. Interface Segregation Principle (ISP)

Objetivo: Clientes não devem ser forçados a depender de interfaces que não utilizam.

Aplicação: Interfaces específicas como IPessoaJuridicaRepository e IParticipacaoSocietariaService em vez de interfaces genéricas grandes:

```
public interface IPessoaJuridicaRepository : IRepository<PessoaJuridica>{
    // Métodos específicos para PessoaJuridica
}
```

4. Dependency Inversion Principle (DIP)

Objetivo: Módulos de alto nível não devem depender de módulos de baixo nível; ambos devem depender de abstrações.

Aplicação: RegistrarEmpresaService depende de interfaces e não de implementações concretas:

```
public class RegistrarEmpresaService : IRegistrarEmpresaService{
    private readonly IPessoaJuridicaRepository _empresaRepository;
    private readonly IContabilidadeAdapter _contabilidadeAdapter;

    public RegistrarEmpresaService(IPessoaJuridicaRepository empresaRepository,
                                   IContabilidadeAdapter contabilidadeAdapter)    {
        _empresaRepository = empresaRepository;
        _contabilidadeAdapter = contabilidadeAdapter;
    }
}
```

Padrões GRASP

1. Creator

Objetivo: Atribuir a responsabilidade de criação de objetos à classe mais apropriada.

Aplicação: O método factory CriarEmpresa na classe PessoaJuridica:

```
public static PessoaJuridica CriarEmpresa(
    // parâmetros
    IParticipacaoSocietariaService participacaoService,
    List<ParticipacaoSocietaria> socios){
    // Validação e criação
    var empresa = new PessoaJuridica(/*...*/);
    // Adiciona os sócios
    foreach (var socio in sociosList)    {
        empresa.AdicionarSocio(socio);
    }
    return empresa;
}
```

2. Information Expert

Objetivo: Atribuir responsabilidades à classe que tem as informações necessárias.

Aplicação: PessoaJuridica gerencia seu quadro societário pois detém estas informações:

```
public class PessoaJuridica : Pessoa{
        private readonly List<ParticipacaoSocietaria> _quadroSocietario = new List<ParticipacaoSocietaria>();
    public void AdicionarSocio(ParticipacaoSocietaria socio)  {
    _quadroSsocietario.Add(socio);
  }

  public void RemoverSocio(Pessoa socio)  {
    _quadroSocietario.RemoveAll(p => p.Socio.Id == socio.Id);
  }
  public decimal TotalParticipacoes() => _quadroSocietario.Sum(p => p.ValorParticipacao);
}
```

3. Low Coupling

Objetivo: Reduzir dependências entre classes para aumentar reusabilidade e diminuir impacto de mudanças.

Aplicação: O uso de interfaces e a implementação do padrão adapter:

```
public class ContabilidadeAdapter : IContabilidadeAdapter{
  private readonly ISistemaContabil _sistemaContabil;

  public void EnviarDadosEmpresa(PessoaJuridica empresa)  {
    _sistemaContabil.RegistrarEmpresa(
      empresa.Cnpj,
      empresa.Nome,
      empresa.CapitalSocial
    );
  }
}
```

4. High Cohesion

Objetivo: Manter as classes focadas, compreensíveis e gerenciáveis.

Aplicação: Cada classe tem uma responsabilidade bem definida e focada:

```
public class ParticipacaoSocietariaService : IParticipacaoSocietariaService{
```

```
    public void ValidarParticipacoes(List<ParticipacaoSocietaria> participacoes, decimal capitalSocial) {
        // Apenas validações relacionadas a participações societárias
    }
}
```