Instituto Politécnico do Cávado e do Ave

Escola Superior de Tecnologia

Programação Orientada a Objetos

Licenciatura em Engenharia de Sistemas Informáticos

Trabalho Prático

Gestão de Condomínios

Fábio Alexandre Gomes Fernandes – a22996

Pedro Lourenço Morais Rocha – a23009

dezembro de 2023

Resumo

O programa de gestão de condomínios desenvolvido como parte da disciplina de Programação Orientada a Objetos é uma expressão concreta dos princípios adquiridos durante o curso. Com foco na eficiência da gestão de condomínios, o sistema aborda a representação e manipulação de elementos-chave, tais como condomínios, proprietários, despesas, reuniões e documentos.

A hierarquia de classes foi estruturada para espelhar a realidade, proporcionando uma visão clara da interconexão entre diferentes entidades. A implementação de regras de negócio e exceções personalizadas contribui para a confiabilidade e integridade do programa, assegurando que apenas dados válidos e consistentes sejam manipulados.

Ao explorar as funcionalidades e estruturas fundamentais do sistema, este relatório destaca o compromisso com uma aplicação eficaz dos conceitos de POO, oferecendo uma valiosa contribuição para a compreensão prática dos princípios ensinados na disciplina.

Índice

Índice de Ilustrações

1. Introdução

No âmbito da cadeira de Programação Orientada a Objetos, lecionada por Luís Ferreira no curso de Engenharia de Sistemas Informáticos, apresentamos este relatório sobre o desenvolvimento de um programa de gestão de condomínios. A gestão eficiente de condomínios é um desafio que envolve a coordenação de diversos elementos, desde a administração de propriedades e despesas até a organização de reuniões e documentação. Com o objetivo de simplificar e otimizar esse processo, desenvolvemos uma solução intuitiva que se alinha aos princípios da Programação Orientada a Objetos. Este sistema não é apenas uma ferramenta tecnológica, mas sim uma aplicação prática dos conceitos aprendidos na disciplina. Ao longo deste relatório, exploraremos as principais funcionalidades e estruturas do programa, destacando como cada componente está interligado para criar uma ferramenta eficaz de gestão condomínios. Desde a representação de diferentes tipos de utilizadores até a hierarquia de classes e relações entre elas, examinaremos de perto as decisões de design que fundamentam a arquitetura do software.

Este trabalho busca não apenas demonstrar a aplicação prática dos conceitos de Programação Orientada a Objetos, mas também oferecer uma solução valiosa para um contexto real, contribuindo para uma compreensão mais aprofundada dos desafios e oportunidades associados à implementação desses princípios.

2. Sistema de Gestão de Condomínios

2.1 Objetos de Negócio

[1]–[5]No contexto de sistemas de gestão de condomínios, a implementação eficiente e organizada de objetos de negócio desempenha um papel crucial. Esses objetos encapsulam as entidades fundamentais do domínio, proporcionando uma representação estruturada e funcional dos elementos centrais do sistema. No âmbito do tópico 3.1, destaca-se os principais objetos de negócio que compõem a arquitetura do sistema, cada um desempenhando um papel específico na gestão e organização de condomínios.

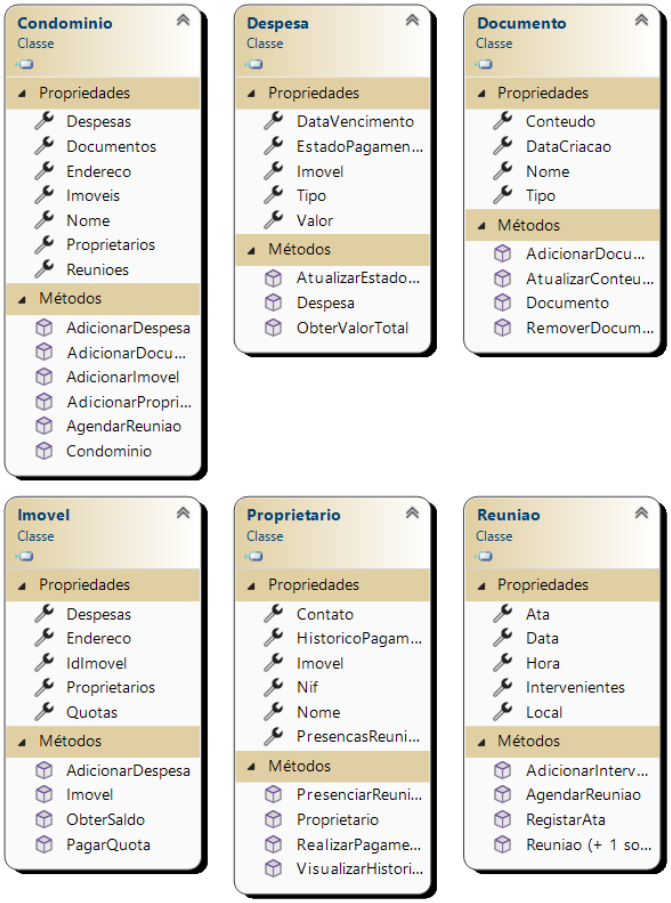

Figura 1: Diagrama dos Objetos de Negócio

2.1.1 Condominio

A classe "*Condominio*" é um componente vital em sistemas de gestão de condomínios. Representa as características de um condomínio, incluindo atributos como nome, endereço e listas associadas a despesas, imóveis, proprietários, reuniões e documentos. Os métodos permitem adicionar informações e a classe é marcada como "*Serializable*". A inicialização de listas como "*null!*" é necessária para garantir que sejam corretamente inicializadas antes de uso. Análise:

Atributos

- **"*nome*":** Representa o nome do condomínio.
- **"*endereco*":** Indica o endereço do condomínio.
- **"*despesas*":** Lista das despesas relacionadas ao condomínio.
- **"*imoveis*":** Lista dos imóveis vinculados ao condomínio.
- **"*proprietários*":** Lista dos proprietários associados ao condomínio.
- **"*reunioes*":** Lista das reuniões agendadas no contexto do condomínio.
- **"*documentos*":** Lista dos documentos ligados ao condomínio.

Métodos

- **Construtores:**
    - **"*Condominio(string nome, string endereco, List<string> despesas, List<string> imoveis, List<string> proprietarios, List<string> reunioes, List<string> documentos)*":** Um construtor que permite a inicialização de um objeto "*Condominio*" com informações específicas. Este construtor aceita parâmetros que representam o nome, endereço, despesas, imóveis, proprietários, reuniões e documentos associados ao condomínio.

```
public Condominio(string nome, string endereco, List<str
{
    Nome = nome;
    Endereco = endereco;
    Despesas = despesas;
    Imoveis = imoveis;
    Proprietarios = proprietarios;
    Reunioes = reunioes;
    Documentos = documentos;
}
```

Figura 2: Objetos de Negócio (Condominio - Construtores)

- **Propriedades:**
  - **"*Nome*":** Propriedade que procura ou define o nome do condomínio.
  - **"*Endereco*":** Propriedade que procura ou define o endereço do condomínio.
  - **"*Despesas*":** Propriedade que procura ou define a lista de despesas associadas ao condomínio.
  - **"*Imoveis*":** Propriedade que procura ou define a lista de imóveis vinculados ao condomínio.
  - **"*Proprietarios*":** Propriedade que procura ou define a lista de proprietários associados ao condomínio.
  - **"*Reunioes*":** Propriedade que procura ou define a lista de reuniões agendadas no condomínio.
  - **"*Documentos*":** Propriedade que procura ou define a lista de documentos vinculados ao condomínio.

- **Outros Métodos:**

  - **"*AdicionarDespesa(string despesa)*":** Adiciona uma nova despesa à lista associada ao condomínio.
  - **"*AdicionarImovel(string imovel)*":** Inclui um novo imóvel na lista de imóveis associados ao condomínio.
  - **"*AdicionarProprietario(string proprietario)*":** Acrescenta um novo proprietário à lista de proprietários vinculados ao condomínio.
  - **"*AgendarReuniao(string reuniao)*":** Agenda uma nova reunião e a adiciona à lista de reuniões do condomínio.
  - **"*AdicionarDocumento(string documento)*":** Insere um novo documento na lista de documentos associados ao condomínio.

A classe é "*Serializable*", permitindo a serialização para armazenamento ou transmissão. Durante a inicialização dos atributos, utiliza-se "*null!*" para proibir nulos, garantindo que estejam sempre inicializados, assegurando a correta funcionalidade do código.

2.1.2 Despesa

A classe "*Despesa*" modela uma despesa associada a um imóvel. Possui atributos como tipo, valor e data de vencimento. Métodos permitem atualizar o estado de pagamento e obter o valor total. A classe é marcada como "*Serializable*" e inclui um método para calcular o saldo total, considerando despesas e quotas pagas. Análise:

Atributos

- **"*tipo*":** Representa o tipo da despesa.
- **"*valor*":** Indica o valor associado à despesa.
- **"*dataVencimento*":** Representa a data de vencimento da despesa.
- **"*estadoPagamento*":** Indica o estado de pagamento da despesa.
- **"*imovel*":** Representa o imóvel associado à despesa.

Métodos

- **Construtores:**
  - **"*Despesa(string tipo, decimal valor, DateTime dataVencimento, bool estadoPagamento, string imovel)*":** Inicializa uma nova instância da classe "*Despesa*" com os parâmetros especificados. Este construtor aceita o tipo da despesa, valor, data de vencimento, estado de pagamento e o imóvel associado à despesa.

```
public Despesa(string tipo, decimal valor, DateTime dataVencimento, bool estadoPagamento, string imovel)
{
    Tipo = tipo;
    Valor = valor;
    DataVencimento = dataVencimento;
    EstadoPagamento = estadoPagamento;
    Imovel = imovel;
}
```

Figura 3: Objetos de Negócio (Despesa - Construtores)

- **Propriedades:**
  - **"*Tipo*":** Propriedade que procura ou define o tipo da despesa.
  - **"*Valor*":** Propriedade que procura ou define o valor da despesa.
  - **"*DataVencimento*":** Propriedade que procura ou define a data de vencimento da despesa.
  - **"*EstadoPagamento*":** Propriedade que procura ou define o estado de pagamento da despesa.
  - **"*Imovel*":** Propriedade que procura ou define o imóvel associado à despesa.

- **Outros Métodos:**
  - **"*AtualizarEstadoPagamento(bool novoEstado)*":** Atualiza o estado de pagamento da despesa com base no novo estado fornecido.
  - **"*ObterValorTotal()*":** Obtém o valor total da despesa.

A classe é "*Serializable*", permitindo a serialização para armazenamento ou transmissão. Durante a inicialização dos atributos, utiliza-se "*null!*" para proibir nulos, garantindo que estejam sempre inicializados, assegurando a correta funcionalidade do código.

2.1.3 Documento

A classe "*Documento*" representa documentos no sistema, com atributos como tipo, data de criação, conteúdo e nome. Métodos permitem adicionar, remover e atualizar documentos. A classe é marcada como "*Serializable*" e inclui métodos para manipulação de conteúdo.

Atributos

- **"*tipo*":** Representa o tipo do documento.
- **"*dataCriacao*":** Indica a data de criação do documento.
- **"*conteudo*":** Contém o conteúdo do documento.
- **"*nome*":** Armazena o nome do documento.

Métodos

- **Construtores**
  - **"*Documento(string tipo, DateTime dataCriacao, string conteudo, string nome)*":** Inicializa uma nova instância da classe "*Documento*" com os parâmetros especificados. Este construtor aceita o tipo do documento, a data de criação, o conteúdo e o nome do documento.

```
public Documento(string tipo, DateTime dataCriacao, string conteudo, string nome)
{
    Tipo = tipo;
    DataCriacao = dataCriacao;
    Conteudo = conteudo;
    Nome = nome;
}
```

Figura 4: Objetos de Negócio (Documento - Construtores)

- **Propriedades:**
  - **"*Tipo*":** Propriedade que procura ou define o tipo do documento.
  - **"*DataCriacao*":** Propriedade que procura ou define a data de criação do documento.
  - **"*Conteudo*":** Propriedade que procura ou define o conteúdo do documento.
  - **"*Nome*":** Propriedade que procura ou define o nome do documento.

- **Outros Métodos:**
  - **"*AdicionarDocumento(string tipo, DateTime dataCriacao, string conteudo, string nome)*":** Adiciona um novo documento com o tipo, data de criação, conteúdo e nome fornecidos.
  - **"*RemoverDocumento()*":** Remove o documento, definindo todos os atributos como vazios ou padrão.
  - **"*AtualizarConteudo(string conteudo)*":** Adiciona ou atualiza o documento com o conteúdo fornecido.

A classe é marcada como "*Serializable*", permitindo a serialização dos objetos para armazenamento ou transmissão.

Na inicialização, os atributos são definidos como "*string.Empty*" e "*DateTime.MinValue*", assegurando que estejam sempre inicializados antes do uso, mantendo a funcionalidade correta do código.

2.1.4 Imovel

A classe "*Imovel*" modela um imóvel, incluindo atributos como ID, proprietários, despesas, quotas e endereço. Métodos permitem adicionar despesas, pagar quotas e obter o saldo total do imóvel. A classe é marcada como "*Serializable*" e inclui métodos para manipulação de informações financeiras. Análise:

Atributos

- **"*idImovel*":** Identificador único do imóvel.
- **"*proprietarios*":** Lista de proprietários do imóvel.
- **"*despesas*":** Lista de despesas associadas ao imóvel.
- **"*quotas*":** Lista de quotas associadas ao imóvel.
- **"*endereço*":** Endereço do imóvel.

Métodos

- **Construtores:**
  - **"*Imovel(int idImovel, List<Proprietario> proprietarios, string endereco)*":** Inicializa uma nova instância da classe "*Imovel*" com os parâmetros especificados. Este construtor aceita o identificador único do imóvel, a lista de proprietários e o endereço do imóvel.

```
public Imovel(int idImovel, List<Proprietario> proprietarios, string endereco)
{
    IdImovel = idImovel;

    // Inicializa as listas de despesas e quotas
    Despesas = new List<Despesa>();
    Quotas = new List<decimal>();

    // Define a lista de proprietários
    Proprietarios = proprietarios;

    // Define o endereço do imóvel
    Endereco = endereco;
}
```

Figura 5: Objetos de Negócio (Imovel - Construtores)

- **Propriedades:**
  - **"*IdImovel*":** Propriedade que procura ou define o identificador único do imóvel.
  - **"*Proprietarios*":** Propriedade que procura ou define a lista de proprietários do imóvel.
  - **"*Despesas*":** Propriedade que procura ou define a lista de despesas associadas ao imóvel.
  - **"*Quotas*":** Propriedade que procura ou define a lista de quotas associadas ao imóvel.
  - **"*Endereco*":** Propriedade que procura ou define o endereço do imóvel.

- **Outros Métodos**
  - **"*AdicionarDespesa(Despesa despesa)*":** Adiciona uma despesa à lista de despesas associadas ao imóvel.
  - **"*PagarQuota(decimal valorPago)*":** Registra o pagamento de uma quota do imóvel.
  - **"*ObterSaldo()*":** Obtém o saldo total do imóvel considerando as despesas e quotas pagas.

A classe é "*Serializable*", permitindo a serialização para armazenamento ou transmissão. Durante a inicialização dos atributos, utiliza-se "*null!*" para proibir nulos, garantindo que estejam sempre inicializados, assegurando a correta funcionalidade do código.

2.1.5 Proprietario

A classe "*Proprietario*" representa proprietários de imóveis. Possui atributos como nome, contato, imóvel associado, NIF e listas de histórico de pagamentos e presenças em reuniões. Métodos incluem realizar pagamentos, visualizar histórico e registar presença em reuniões. Análise:

Atributos

- **"*nome*":** Nome do proprietário.
- **"*contato*":** Contato do proprietário.
- **"*imovel*":** Imóvel associado ao proprietário.
- **"*nif*":** NIF (Número de Identificação Fiscal) do proprietário.
- **"*historicoPagamentos*":** Lista de histórico de pagamentos do proprietário.
- **"*presencasReunioes*":** Lista de presenças em reuniões do proprietário.

Métodos

- **Construtores:**
  - **"*Proprietario(string nome, string contato, string imovel, string nif)*":** Construtor predefinido que inicializa uma nova instância da classe "*Proprietario*" com os parâmetros especificados. Este construtor aceita o nome, contato, imóvel e NIF do proprietário, inicializando também as listas de histórico de pagamentos e presenças em reuniões.

```
public Proprietario(string nome, string contato, string imovel, string nif)
{
    Nome = nome;
    Contato = contato;
    Imovel = imovel;
    Nif = nif;
    HistoricoPagamentos = new List<string>();
    PresencasReunioes = new List<string>();
}
```

Figura 6: Objetos de Negócio (Proprietario - Construtores)

- **Propriedades:**
  - **"*Nome*":** Propriedade que procura ou define o nome do proprietário.
  - **"*Contato*":** Propriedade que procura ou define o contato do proprietário.
  - **"*Imovel*":** Propriedade que procura ou define o imóvel associado ao proprietário.
  - **"*Nif*":** Propriedade que procura ou define o NIF do proprietário.
  - **"*HistoricoPagamentos*":** Propriedade que obtém a lista de histórico de pagamentos do proprietário.
  - **"*PresencasReunioes*":** Propriedade que obtém a lista de presenças em reuniões do proprietário.

- **Outros Métodos:**
  - **"*RealizarPagamento(decimal valor)*":** Realiza o pagamento do proprietário com o valor especificado, registrando no histórico de pagamentos.
  - **"*VisualizarHistorico()*":** Visualiza o histórico de pagamentos do proprietário.
  - **"*PresenciarReuniao()*":** Regista a presença do proprietário em uma reunião, retornando uma mensagem que indica a presença.

A classe é "*Serializable*", permitindo a serialização para armazenamento ou transmissão. Durante a inicialização dos atributos, utiliza-se "*null!*" para proibir nulos, garantindo que estejam sempre inicializados, assegurando a correta funcionalidade do código.

2.1.6 Reuniao

A classe "*Reuniao*" modela reuniões, incluindo atributos como data, hora, local, intervenientes e ata. Métodos permitem agendar reuniões, adicionar intervenientes e registar a ata. A classe é marcada como "*Serializable*" e fornece funcionalidades essenciais para o gerenciamento de reuniões condóminas. Análise:

Atributos

- **"*data*":** Representa a data da reunião.
- **"*hora*":** Representa a hora da reunião.
- **"*local*":** Representa o local da reunião.
- **"*intervenientes*":** Lista de intervenientes na reunião.
- **"*ata*":** Representa a ata da reunião.

Métodos

- **Construtores:**
  - **"*Reuniao()*":** Construtor padrão que inicializa uma nova instância da classe "*Reuniao*", inicializando a lista de intervenientes.
  - **"*Reuniao(DateTime data, TimeSpan hora, string local)*":** Inicializa uma nova instância da classe "*Reuniao*" com os parâmetros especificados. Este construtor chama o construtor padrão para garantir que a lista de intervenientes seja inicializada.

```
#region Construtores

/// <summary> Construtor padrão da classe Reuniao.
1 referência
public Reuniao()
{
    intervenientes = new List<string>();
}

/// <summary> Inicializa uma nova instância da classe Reuniao com os parâmetros  ...
0 referências
public Reuniao(DateTime data, TimeSpan hora, string local) : this()
{
    Data = data;
    Hora = hora;
    Local = local;
}

#endregion
```

Figura 7: Objetos de Negócio (Reuniao - Construtores)

- **Propriedades:**
  - **"*Data*":** Propriedade que procura ou define a data da reunião.
  - **"*Hora*":** Propriedade que procura ou define a hora da reunião.
  - **"*Local*":** Propriedade que procura ou define o local da reunião.
  - **"*Intervenientes*":** Propriedade que procura ou define a lista de intervenientes na reunião.
  - **"*Ata*":** Propriedade que procura ou define a ata da reunião.

- **Outros Métodos:**
  - **"*AgendarReuniao(DateTime data, TimeSpan hora, string local)*":** Agenda a reunião com a data, hora e local especificados.
  - **"*AdicionarInterveniente(string interveniente)*":** Adiciona um interveniente à lista de intervenientes na reunião.
  - **"*RegistarAta(string ata)*":** Registra a ata da reunião.

A classe é marcada como "*Serializable*", permitindo a serialização dos objetos para armazenamento ou transmissão. Na inicialização, os atributos são definidos como "*string.Empty*", assegurando que estejam sempre inicializados antes do uso, mantendo a funcionalidade correta do código.

2.2 Dados

[6], [7]O tópico 3.2 aborda a componente crucial de dados no contexto do sistema, com enfoque em diversas entidades-chave. As classes apresentadas, nomeadamente "*Condominios*", "*Despesas*", "*Documentos*", "*Imoveis*", "*Proprietarios*" e "*Reunioes*" desempenham papéis essenciais na organização e manipulação de informações relevantes para o sistema em questão.

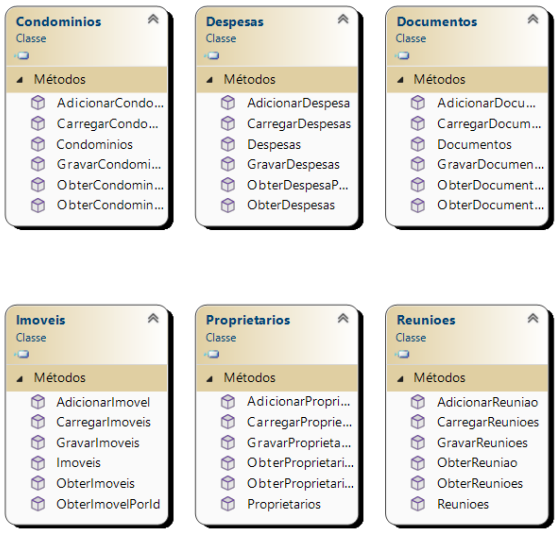

Figura 8: Diagrama dos Dados

2.2.1 Condominios

A classe "*Condominios"* é responsável por armazenar e gerenciar informações sobre condomínios. Vamos analisar detalhadamente a estrutura da classe:

Estrutura da Classe:

1. **Declarações de Bibliotecas**
   - **"*ObjetosNegocio*":** "*DLL*" criada através dos objetos de negócios.
   - **"*Excecoes*":** "*DLL*" criada através das exceções.
   - **"*RegrasNegocio*":** "*DLL*" criada através das regras de negócio.

2. **Atributos**
   - **"*private List<Condominio> listaCondominios*":** Lista privada que armazena objetos do tipo "*Condominio*".

3. **Construtores**
   - **"*public Condominios()*":** Construtor padrão que inicializa a lista de condomínios ao criar uma nova instância da classe.

4. **Métodos**
   - **"*AdicionarCondominio*":** Método público que adiciona um novo condomínio à lista. Pode lançar uma exceção personalizada em caso de duplicidade. Valida se o condomínio já existe na lista antes de adicioná-lo.

```
public void AdicionarCondominio(Condominio condominio, bool lancaExcecao = false)
{
    if (!listaCondominios.Contains(condominio))
    {
        if (lancaExcecao)
        {
            throw new CondominioException.CondominioDuplicadoException(condominio.Nome, condominio.Endereco);
        }
    }
    else
    {
        CondominioRegras.ValidarCondominio(condominio);
        listaCondominios.Add(condominio);
    }
}
```

Figura 9: Dados (Condominios - AdicionarCondominio)

   - **"*ObterCondominios*":** Método público que retorna a lista completa de condomínios armazenados.
   - **"*ObterCondominioPorNome*":** Método público que recebe o nome como parâmetro e retorna o condomínio correspondente ou "*null*" se não encontrado. Utiliza o método "*Find*" da lista.

- **"*GravarCondominios*":** Método público que grava a lista de condomínios em um arquivo usando serialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.
- **"*CarregarCondominios*":** Método público que carrega a lista de condomínios a partir de um arquivo usando desserialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.

```
/// <summary> Grava a lista de condomínios em um arquivo usando serialização JSO ...
0 referências
public void GravarCondominios(string caminhoArquivo)
{
    try
    {
        string jsonString = JsonSerializer.Serialize(listaCondominios);
        File.WriteAllText(caminhoArquivo, jsonString);
    }
    catch (Exception ex)
    {
        throw new CondominioException.GravarCondominiosException(ex.Message);
    }
}

/// <summary> Carrega a lista de condomínios a partir de um arquivo usando desse ...
0 referências
public void CarregarCondominios(string caminhoArquivo)
{
    try
    {
        string jsonString = File.ReadAllText(caminhoArquivo);
        listaCondominios = JsonSerializer.Deserialize<List<Condominio>>(jsonString)!;
    }
    catch (Exception ex)
    {
        throw new CondominioException.CarregarCondominiosException(ex.Message);
    }
}
```

Figura 10: Dados (Condominios - GravarCondominios e CarregarCondominios)

2.2.2 Despesas

A classe "*Despesas*" é responsável por armazenar e gerenciar informações sobre despesas. Vamos analisar detalhadamente a estrutura da classe:

Estrutura da Classe:

1. **Declarações de Bibliotecas**
   - **"*ObjetosNegocio*":** "*DLL*" criada através dos objetos de negócios.
   - **"*Excecoes*":** "*DLL*" criada através das exceções.
   - **"*RegrasNegocio*":** "*DLL*" criada através das regras de negócio.

2. **Atributos**
   - **"*private List<Despesa> listaDespesas*":** Lista privada que armazena objetos do tipo "*Despesa*".

3. **Construtores**
   - **"*public Despesas()*":** Construtor padrão que inicializa a lista de despesas ao criar uma nova instância da classe.

4. **Métodos**
   - **"*AdicionarDespesa*":** Método público que adiciona uma nova despesa à lista. Pode lançar uma exceção personalizada em caso de duplicidade. Valida se a despesa já existe na lista antes de adicioná-la.
   - **"*ObterDespesas*":** Método público que retorna a lista completa de despesas armazenadas.
   - **"*ObterDespesaPorTipoValorEData*":** Método público que recebe tipo, valor e data de vencimento como parâmetros e retorna a despesa correspondente ou "*null*" se não encontrada. Utiliza o método "*Find*" da lista.
   - **"*GravarDespesas*":** Método público que grava a lista de despesas em um arquivo usando serialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.
   - **"*CarregarDespesas*":** Método público que carrega a lista de despesas a partir de um arquivo usando desserialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.

2.2.3 Documentos

A classe "*Documentos*" é responsável por armazenar e gerenciar informações sobre documentos. Vamos analisar detalhadamente a estrutura da classe:

Estrutura da Classe:

1. **Declarações de Bibliotecas**
   - **"*ObjetosNegocio*":** "*DLL*" criada através dos objetos de negócios.
   - **"*Excecoes*":** "*DLL*" criada através das exceções.
   - **"*RegrasNegocio*":** "*DLL*" criada através das regras de negócio.

2. **Atributos**
   - **"*private List<Documento> listaDocumentos*":** Lista privada que armazena objetos do tipo "*Documento*".

3. **Construtores**
   - **"*public Documentos()*":** Construtor padrão que inicializa a lista de documentos ao criar uma nova instância da classe.

4. **Métodos**
   - **"AdicionarDocumento":** Método público que adiciona um novo documento à lista. Pode lançar uma exceção personalizada em caso de duplicidade. Valida se o documento já existe na lista antes de adicioná-lo.
   - **"ObterDocumentos":** Método público que retorna a lista completa de documentos armazenados.
   - **"ObterDocumentoPorTipoENome":** Método público que recebe tipo e nome como parâmetros e retorna o documento correspondente ou "*null*" se não encontrado. Utiliza o método "*Find*" da lista.
   - **"GravarDocumentos":** Método público que grava a lista de documentos em um arquivo usando serialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.
   - **"CarregarDocumentos":** Método público que carrega a lista de documentos a partir de um arquivo usando desserialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.

2.2.4 Imoveis

A classe "*Imoveis*" é responsável por armazenar e gerenciar informações relacionadas a imóveis. Vamos analisar detalhadamente a estrutura da classe:

Estrutura da Classe:

1. **Declarações de Bibliotecas**
   - **"*ObjetosNegocio*":** "*DLL*" criada através dos objetos de negócios.
   - **"*Excecoes*":** "*DLL*" criada através das exceções.
   - **"*RegrasNegocio*":** "*DLL*" criada através das regras de negócio.

2. **Atributos**
   - **"*private List<Imovel> listaImoveis*":** Lista privada que armazena objetos do tipo "*Imovel*".

3. **Construtores**
   - **"*public Imoveis()*":** Construtor padrão que inicializa a lista de imóveis ao criar uma nova instância da classe.

4. **Métodos**
   - **"*AdicionarImovel*":** Método público que adiciona um novo imóvel à lista. Pode lançar uma exceção personalizada em caso de duplicidade. Valida se o imóvel já existe na lista antes de adicioná-lo.
   - **"*ObterImoveis*":** Método público que retorna a lista completa de imóveis armazenados.
   - **"*ObterImovelPorId*":** Método público que recebe o identificador único como parâmetro e retorna o imóvel correspondente ou "*null*" se não encontrado. Utiliza o método "*Find*" da lista.
   - **"*GravarImoveis*":** Método público que grava a lista de imóveis em um arquivo usando serialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.

- **"*CarregarImoveis*":** Método público que carrega a lista de imóveis a partir de um arquivo usando desserialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.

2.2.5 Proprietarios

A classe "Proprietarios" é responsável por armazenar e gerenciar informações relacionadas a proprietários. Vamos analisar detalhadamente a estrutura da classe:

Estrutura da Classe:

1. **Declarações de Bibliotecas**
   - **"*ObjetosNegocio*":** "*DLL*" criada através dos objetos de negócios.
   - **"*Excecoes*":** "*DLL*" criada através das exceções.
   - **"*RegrasNegocio*":** "*DLL*" criada através das regras de negócio.

2. **Atributos**
   - **"*private List<Proprietario> listaProprietarios*":** Lista privada que armazena objetos do tipo "*Proprietario*".

3. **Construtores**
   - **"*public Proprietarios()*":** Construtor padrão que inicializa a lista de proprietários ao criar uma nova instância da classe.

4. **Métodos**
   - **"*AdicionarProprietario*":** Método público que adiciona um novo proprietário à lista. Pode lançar uma exceção personalizada em caso de duplicidade. Valida se o proprietário já existe na lista antes de adicioná-lo.
   - **"*ObterProprietarios*":** Método público que retorna a lista completa de proprietários armazenados.

- **"*ObterProprietarioPorNif*":** Método público que recebe o NIF como parâmetro e retorna o proprietário correspondente ou "*null*" se não encontrado. Utiliza o método "*Find*" da lista.
- **"*GravarProprietarios*":** Método público que grava a lista de proprietários em um arquivo usando serialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.
- **"*CarregarProprietarios*":** Método público que carrega a lista de proprietários a partir de um arquivo usando desserialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.

2.2.6 Reunioes

A classe "Reunioes" é responsável por armazenar e gerenciar informações relacionadas a reuniões. Vamos analisar detalhadamente a estrutura da classe:

Estrutura da Classe:

1. **Declarações de Bibliotecas (Linhas 1-7)**
   - **"*ObjetosNegocio*":** "*DLL*" criada através dos objetos de negócios.
   - **"*Excecoes*":** "*DLL*" criada através das exceções.
   - **"*RegrasNegocio*":** "*DLL*" criada através das regras de negócio.

2. **Atributos**
   - **"*private List<Reuniao> listaReunioes*":** Lista privada que armazena objetos do tipo "*Reuniao*".

3. **Construtores**
   - **"*public Reunioes()*":** Construtor padrão que inicializa a lista de reuniões ao criar uma nova instância da classe.

4. **Métodos**
   - **"*AdicionarReuniao*":** Método público que adiciona uma nova reunião à lista. Pode lançar uma exceção personalizada em caso de duplicidade. Valida se a reunião já existe na lista antes de adicioná-la.
   - **"*ObterReunioes*":** Método público que retorna a lista completa de reuniões armazenadas.
   - **"*ObterReuniao*":** Método público que recebe a data e a hora como parâmetros e retorna a reunião correspondente ou "*null*" se não encontrada. Utiliza o método "*Find*" da lista.
   - **"*GravarReunioes*":** Método público que grava a lista de reuniões em um arquivo usando serialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.
   - **"*CarregarReunioes*":** Método público que carrega a lista de reuniões a partir de um arquivo usando desserialização "*JSON*". Lança uma exceção personalizada em caso de erro durante o processo.

2.3 Exceções

No contexto deste projeto, a estrutura de exceções foi organizada em distintos domínios, cada um correspondendo a uma entidade específica do sistema. Essa divisão facilita a identificação e o tratamento de erros, proporcionando uma estrutura coesa e clara no código. As exceções foram distribuídas nos seguintes domínios:

1. *CondominioException:* Exceções específicas de condomínios.
2. *DespesaException:* Exceções específicas de despesas.
3. *DocumentoException:* Exceções específicas de documentos.
4. *ImovelException:* Exceções específicas de imóveis.
5. *ProprietarioException:* Exceções específicas de proprietários.
6. *ReuniaoException:* Exceções específicas de reuniões.

Essa abordagem visa simplificar a identificação e o tratamento de erros, proporcionando uma implementação robusta e de fácil manutenção no sistema.

Figura 11: Exceções (Diagrama de ImovelException)

2.3.1 CondominioException

O domínio "*CondominioException*" engloba exceções relacionadas ao contexto de condomínios no sistema. Aqui estão as explicações detalhadas para cada exceção neste domínio:

***"CondominioException"* (Exceção Base):**

Descrição: Classe base para exceções relacionadas ao domínio de condomínios.

Propósito: Permite capturar erros genéricos associados a operações com entidades de condomínios.

Exemplo de Uso: Útil para identificar e tratar problemas que não se enquadram em categorias mais específicas de exceções em operações envolvendo condomínios.

```
42 referências
public class CondominioException : Exception
{
    10 referências
    public CondominioException(string message) ...

    /// <summary> Exceção lançada quando o nome do condomínio é nulo ou vazio.
    3 referências
    public class NomeCondominioNuloOuVazioException ...

    /// <summary> Exceção lançada quando o endereço do condomínio é nulo ou vazio.
    3 referências
    public class EnderecoCondominioNuloOuVazioException ...

    /// <summary> Exceção lançada quando a lista de despesas associadas ao condomíni ...
    3 referências
    public class DespesasCondominioVaziasException ...

    /// <summary> Exceção lançada quando a lista de imóveis associados ao condomínio ...
    3 referências
    public class ImoveisCondominioVaziosException ...

    /// <summary> Exceção lançada quando a lista de proprietários associados ao cond ...
    3 referências
    public class ProprietariosCondominioVaziosException ...

    /// <summary> Exceção lançada quando a lista de reuniões agendadas no condomínio ...
    3 referências
    public class ReunioesCondominioVaziasException ...

    /// <summary> Exceção lançada quando a lista de documentos associados ao condomí ...
    3 referências
    public class DocumentosCondominioVaziosException ...

    /// <summary> Exceção lançada quando um condomínio duplicado é adicionado à list ...
    3 referências
    public class CondominioDuplicadoException ...

    /// <summary> Exceção lançada ao ocorrer um erro durante a gravação de condomíni ...
    3 referências
    public class GravarCondominiosException ...

    /// <summary> Exceção lançada ao ocorrer um erro durante o carregamento de condo ...
    3 referências
    public class CarregarCondominiosException ...
}
```

Figura 12: Exceções (CondominioException)

"***NomeCondominioNuloOuVazioException***":

Descrição: Lançada quando o nome do condomínio é nulo ou vazio.

Propósito: Assegura que o nome do condomínio seja sempre fornecido, evitando a criação de condomínios sem nome.

Exemplo de Uso: Garante que cada condomínio tenha um nome associado, impedindo a criação de condomínios sem essa informação.

"***EnderecoCondominioNuloOuVazioException***":

Descrição: Lançada quando o endereço do condomínio é nulo ou vazio.

Propósito: Assegura que o endereço do condomínio seja sempre fornecido, evitando condomínios sem endereço.

Exemplo de Uso: Garante que cada condomínio tenha um endereço associado, impedindo a criação de condomínios sem essa informação.

"*DespesasCondominioVaziasException*":

Descrição: Lançada quando a lista de despesas associadas ao condomínio está vazia.

Propósito: Assegura que haja pelo menos uma despesa associada ao condomínio, evitando condomínios sem despesas.

Exemplo de Uso: Garante que um condomínio tenha despesas associadas, impedindo a criação de condomínios sem essa informação.

"*ImoveisCondominioVaziosException*":

Descrição: Lançada quando a lista de imóveis associados ao condomínio está vazia.

Propósito: Assegura que haja pelo menos um imóvel associado ao condomínio, evitando condomínios sem imóveis.

Exemplo de Uso: Garante que um condomínio tenha imóveis associados, impedindo a criação de condomínios sem essa informação.

"*ProprietariosCondominioVaziosException*":

Descrição: Lançada quando a lista de proprietários associados ao condomínio está vazia.

Propósito: Assegura que haja pelo menos um proprietário associado ao condomínio, evitando condomínios sem proprietários.

Exemplo de Uso: Garante que um condomínio tenha proprietários associados, impedindo a criação de condomínios sem essa informação.

"*ReunioesCondominioVaziasException*":

Descrição: Lançada quando a lista de reuniões agendadas no condomínio está vazia.

Propósito: Assegura que haja pelo menos uma reunião agendada no condomínio, evitando condomínios sem reuniões.

Exemplo de Uso: Garante que um condomínio tenha reuniões agendadas, impedindo a criação de condomínios sem essa informação.

"*DocumentosCondominioVaziosException*":

Descrição: Lançada quando a lista de documentos associados ao condomínio está vazia.

Propósito: Assegura que haja pelo menos um documento associado ao condomínio, evitando condomínios sem documentos.

Exemplo de Uso: Garante que um condomínio tenha documentos associados, impedindo a criação de condomínios sem essa informação.

"*CondominioDuplicadoException*":

Descrição: Lançada quando um condomínio duplicado é adicionado à lista.

Propósito: Evita a adição de condomínios com o mesmo nome e endereço, garantindo a unicidade na lista.

Exemplo de Uso: Impede a inclusão de condomínios duplicados na lista.

"*GravarCondominiosException*":

Descrição: Lançada ao ocorrer um erro durante a gravação de condomínios.

Propósito: Facilita a identificação e tratamento de problemas ao salvar dados de condomínios, fornecendo informações específicas sobre o erro ocorrido.

Exemplo de Uso: Ajuda a lidar com falhas durante a persistência de dados de condomínios.

"*CarregarCondominiosException*":

Descrição: Lançada ao ocorrer um erro durante o carregamento de condomínios.

Propósito: Ajuda na identificação e resolução de problemas ao carregar dados de condomínios, oferecendo detalhes sobre o erro ocorrido.

Exemplo de Uso: Facilita a depuração de problemas durante o processo de carregamento de dados de condomínios.

2.3.2 DespesaException

O domínio "*DespesaException*" é responsável por gerenciar exceções relacionadas ao contexto de despesas no sistema. Aqui estão as explicações para cada exceção dentro deste domínio:

"***DespesaException (Exceção Base)***":

Descrição: Classe base para exceções relacionadas ao domínio de Despesa.

Propósito: Permite capturar erros genéricos associados a operações com entidades de despesas.

Exemplo de Uso: Pode ser útil para identificar e tratar problemas que não se enquadram em categorias mais específicas.

"***TipoDespesaNuloOuVazioException***":

Descrição: Lançada quando o tipo da despesa é nulo ou vazio.

Propósito: Assegura que o tipo da despesa seja sempre fornecido, evitando a criação de despesas sem tipo.

Exemplo de Uso: Garante que cada despesa tenha um tipo associado, impedindo a criação de despesas sem essa informação.

"***ImovelDespesaNuloOuVazioException***":

Descrição: Lançada quando o imóvel associado à despesa é nulo ou vazio.

Propósito: Certifica-se de que haja um imóvel associado à despesa, impedindo despesas sem associação a um imóvel específico.

Exemplo de Uso: Garante que cada despesa esteja vinculada a um imóvel, evitando despesas sem essa informação essencial.

"***ValorDespesaInvalidoException***":

Descrição: Lançada quando o valor da despesa é inválido (deve ser maior que zero).

Propósito: Garante que o valor da despesa seja sempre válido, evitando despesas com valores não permitidos.

Exemplo de Uso: Impede a criação de despesas com valores não positivos, garantindo consistência nos dados.

"***DataVencimentoDespesaPassadaException***":

Descrição: Lançada quando a data de vencimento da despesa é no passado.

Propósito: Certifica-se de que a data de vencimento esteja no futuro, evitando despesas com datas expiradas.

Exemplo de Uso: Garante que as despesas tenham datas de vencimento futuras, prevenindo erros relacionados a datas passadas.

"*DespesaDuplicadaException*":

Descrição: Lançada quando uma despesa duplicada é adicionada à lista.

Propósito: Evita a adição de despesas com o mesmo tipo, valor e data de vencimento, mantendo a integridade dos dados.

Exemplo de Uso: Impede a inclusão de despesas duplicadas, garantindo que cada despesa seja única na lista.

"*GravarDespesasException*":

Descrição: Lançada ao ocorrer um erro durante a gravação de despesas.

Propósito: Facilita a identificação e tratamento de problemas ao salvar dados de despesas, fornecendo informações específicas sobre o erro ocorrido.

Exemplo de Uso: Permite aos desenvolvedores lidar de forma adequada com falhas durante a persistência de dados de despesas.

"*CarregarDespesasException*":

Descrição: Lançada ao ocorrer um erro durante o carregamento de despesas.

Propósito: Ajuda na identificação e resolução de problemas ao carregar dados de despesas, oferecendo detalhes sobre o erro ocorrido.

Exemplo de Uso: Facilita a depuração de problemas durante o processo de carregamento de dados de despesas.

2.3.3 DocumentoException

O domínio "*DocumentoException*" é responsável por gerenciar exceções relacionadas ao contexto de documentos no sistema. Aqui estão as explicações detalhadas para cada exceção neste domínio:

"*DocumentoException (Exceção Base)*":

Descrição: Classe base para exceções relacionadas ao domínio de documentos.

Propósito: Permite capturar erros genéricos associados a operações com entidades de documentos.

Exemplo de Uso: Útil para identificar e tratar problemas que não se enquadram em categorias mais específicas de exceções.

"*TipoDocumentoNuloOuVazioException*":

Descrição: Lançada quando o tipo do documento é nulo ou vazio.

Propósito: Assegura que o tipo do documento seja sempre fornecido, evitando a criação de documentos sem tipo.

Exemplo de Uso: Garante que cada documento tenha um tipo associado, impedindo a criação de documentos sem essa informação.

"*NomeDocumentoNuloOuVazioException*":

Descrição: Lançada quando o nome do documento é nulo ou vazio.

Propósito: Assegura que o nome do documento seja sempre fornecido, evitando documentos sem nome.

Exemplo de Uso: Garante que cada documento tenha um nome associado, impedindo a criação de documentos sem essa informação.

"*ConteudoDocumentoNuloOuVazioException*":

Descrição: Lançada quando o conteúdo do documento é nulo ou vazio.

Propósito: Certifica-se de que haja um conteúdo associado ao documento, impedindo documentos sem conteúdo.

Exemplo de Uso: Garante que cada documento tenha conteúdo, evitando a criação de documentos vazios.

"*DocumentoDuplicadoException*":

Descrição: Lançada quando um documento duplicado é adicionado à lista.

Propósito: Evita a adição de documentos com o mesmo nome, garantindo a unicidade na lista.

Exemplo de Uso: Impede a inclusão de documentos duplicados na lista.

"*GravarDocumentosException*":

Descrição: Lançada ao ocorrer um erro durante a gravação de documentos.

Propósito: Facilita a identificação e tratamento de problemas ao salvar dados de documentos, fornecendo informações específicas sobre o erro ocorrido.

Exemplo de Uso: Ajuda a lidar com falhas durante a persistência de dados de documentos.

"*CarregarDocumentosException*":

Descrição: Lançada ao ocorrer um erro durante o carregamento de documentos.

Propósito: Ajuda na identificação e resolução de problemas ao carregar dados de documentos, oferecendo detalhes sobre o erro ocorrido.

Exemplo de Uso: Facilita a depuração de problemas durante o processo de carregamento de dados de documentos.

2.3.4 ImovelException

O domínio "*ImovelException*" abrange exceções relacionadas ao contexto de imóveis no sistema. Abaixo estão explicações detalhadas para cada exceção neste domínio:

"*ImovelException (Exceção Base)*":

Descrição: Classe base para exceções relacionadas ao domínio de imóveis.

Propósito: Permite capturar erros genéricos associados a operações com entidades de imóveis.

Exemplo de Uso: Útil para identificar e tratar problemas que não se enquadram em categorias mais específicas de exceções em operações envolvendo imóveis.

"*DespesasImovelVaziasException*":

Descrição: Lançada quando a lista de despesas associadas ao imóvel está vazia.

Propósito: Garante que um imóvel tenha despesas associadas, evitando a criação de imóveis sem informações sobre despesas.

Exemplo de Uso: Certifica-se de que cada imóvel tenha despesas registadas, impedindo a criação de imóveis sem essa informação.

"*ProprietariosImovelVaziosException*":

Descrição: Lançada quando a lista de proprietários associados ao imóvel está vazia.

Propósito: Garante que um imóvel tenha proprietários associados, evitando a criação de imóveis sem informações sobre proprietários.

Exemplo de Uso: Certifica-se de que cada imóvel tenha proprietários registados, impedindo a criação de imóveis sem essa informação.

"*QuotasImovelVaziasException*":

Descrição: Lançada quando a lista de quotas associadas ao imóvel está vazia.

Propósito: Garante que um imóvel tenha quotas associadas, evitando a criação de imóveis sem informações sobre quotas.

Exemplo de Uso: Certifica-se de que cada imóvel tenha quotas registadas, impedindo a criação de imóveis sem essa informação.

"*EnderecoImovelNuloOuVazioException*":

Descrição: Lançada quando o endereço do imóvel é nulo ou vazio.

Propósito: Certifica-se de que haja um endereço associado ao imóvel, impedindo a criação de imóveis sem essa informação.

Exemplo de Uso: Garante que cada imóvel tenha um endereço associado, impedindo a criação de imóveis sem essa informação.

"*ImovelDuplicadoException*":

Descrição: Lançada quando um imóvel duplicado é adicionado à lista.

Propósito: Evita a adição de imóveis com o mesmo ID, garantindo a unicidade na lista.

Exemplo de Uso: Impede a inclusão de imóveis duplicados na lista.

"*GravarImoveisException*":

Descrição: Lançada ao ocorrer um erro durante a gravação de imóveis.

Propósito: Facilita a identificação e tratamento de problemas ao salvar dados de imóveis, fornecendo informações específicas sobre o erro ocorrido.

Exemplo de Uso: Ajuda a lidar com falhas durante a persistência de dados de imóveis.

“*CarregarImoveisException*”:

Descrição: Lançada ao ocorrer um erro durante o carregamento de imóveis.

Propósito: Ajuda na identificação e resolução de problemas ao carregar dados de imóveis, oferecendo detalhes sobre o erro ocorrido.

Exemplo de Uso: Facilita a depuração de problemas durante o processo de carregamento de dados de imóveis.

2.3.5 ProprietarioException

O domínio “*ProprietarioException*” trata de exceções relacionadas aos proprietários no sistema. Aqui estão as explicações detalhadas para cada exceção neste domínio:

“*ProprietarioException (Exceção Base)*”:

Descrição: Classe base para exceções relacionadas ao domínio de proprietários.

Propósito: Permite capturar erros genéricos associados a operações com entidades de proprietários.

Exemplo de Uso: Útil para identificar e tratar problemas que não se enquadram em categorias mais específicas de exceções em operações envolvendo proprietários.

“*NomeProprietarioNuloOuVazioException*”:

Descrição: Lançada quando o nome do proprietário é nulo ou vazio.

Propósito: Assegura que o nome do proprietário seja sempre fornecido, evitando a criação de proprietários sem nome.

Exemplo de Uso: Garante que cada proprietário tenha um nome associado, impedindo a criação de proprietários sem essa informação.

“*ContatoProprietarioNuloOuVazioException*”:

Descrição: Lançada quando o contato do proprietário é nulo ou vazio.

Propósito: Assegura que o contato do proprietário seja sempre fornecido, evitando proprietários sem informação de contato.

Exemplo de Uso: Garante que cada proprietário tenha um meio de contato associado, impedindo a criação de proprietários sem essa informação.

"*ImovelProprietarioNuloOuVazioException*":

Descrição: Lançada quando o imóvel associado ao proprietário é nulo ou vazio.

Propósito: Assegura que haja pelo menos um imóvel associado ao proprietário, evitando proprietários sem imóveis.

Exemplo de Uso: Garante que cada proprietário esteja associado a pelo menos um imóvel, impedindo a criação de proprietários sem essa informação.

"*NifProprietarioNuloOuVazioException*":

Descrição: Lançada quando o NIF do proprietário é nulo ou vazio.

Propósito: Assegura que o NIF do proprietário seja sempre fornecido, evitando proprietários sem número de identificação fiscal.

Exemplo de Uso: Garante que cada proprietário tenha um NIF associado, impedindo a criação de proprietários sem essa informação.

"*ProprietarioDuplicadoException*":

Descrição: Lançada quando um proprietário duplicado é adicionado à lista.

Propósito: Evita a adição de proprietários com o mesmo NIF, garantindo a unicidade na lista.

Exemplo de Uso: Impede a inclusão de proprietários duplicados na lista.

"*GravarProprietariosException*":

Descrição: Lançada ao ocorrer um erro durante a gravação de proprietários.

Propósito: Facilita a identificação e tratamento de problemas ao salvar dados de proprietários, fornecendo informações específicas sobre o erro ocorrido.

Exemplo de Uso: Ajuda a lidar com falhas durante a persistência de dados de proprietários.

"*CarregarProprietariosException*":

Descrição: Lançada ao ocorrer um erro durante o carregamento de proprietários.

Propósito: Ajuda na identificação e resolução de problemas ao carregar dados de proprietários, oferecendo detalhes sobre o erro ocorrido.

Exemplo de Uso: Facilita a depuração de problemas durante o processo de carregamento de dados de proprietários.

2.3.6 ReuniaoException

O domínio "*ReuniaoException*" é responsável por exceções relacionadas às reuniões no sistema. A seguir estão explicações detalhadas para cada exceção neste domínio:

"*ReuniaoException (Exceção Base)*":

Descrição: Classe base para exceções relacionadas ao domínio de reuniões.

Propósito: Permite capturar erros genéricos associados a operações com entidades de reuniões.

Exemplo de Uso: Útil para identificar e tratar problemas que não se enquadram em categorias mais específicas de exceções em operações envolvendo reuniões.

"*DataReuniaoInvalidaException*":

Descrição: Lançada quando a data da reunião está no passado.

Propósito: Garante que as reuniões só podem ser agendadas para datas futuras.

Exemplo de Uso: Impede a criação de reuniões com datas que já passaram.

"*HoraReuniaoInvalidaException*":

Descrição: Lançada quando a hora da reunião é inválida.

Propósito: Assegura que a hora da reunião esteja dentro de limites válidos.

Exemplo de Uso: Evita a criação de reuniões com horas inválidas.

"*LocalReuniaoNuloOuVazioException*":

Descrição: Lançada quando o local da reunião é nulo ou vazio.

Propósito: Garante que o local da reunião seja sempre fornecido.

Exemplo de Uso: Impede a criação de reuniões sem local.

"*IntervenientesReuniaoVaziosException*":

Descrição: Lançada quando a lista de intervenientes na reunião está vazia.

Propósito: Assegura que haja pelo menos um interveniente na reunião.

Exemplo de Uso: Evita a criação de reuniões sem participantes.

"*ReuniaoDuplicadaException*":

Descrição: Lançada quando uma reunião duplicada é adicionada à lista.

Propósito: Evita a adição de reuniões duplicadas com a mesma data e hora.

Exemplo de Uso: Impede a inclusão de reuniões duplicadas na lista.

"*GravarReunioesException*":

Descrição: Lançada ao ocorrer um erro durante a gravação de reuniões.

Propósito: Facilita a identificação e tratamento de problemas ao salvar dados de reuniões, fornecendo informações específicas sobre o erro ocorrido.

Exemplo de Uso: Ajuda a lidar com falhas durante a persistência de dados de reuniões.

"*CarregarReunioesException*":

Descrição: Lançada ao ocorrer um erro durante o carregamento de reuniões.

Propósito: Ajuda na identificação e resolução de problemas ao carregar dados de reuniões, oferecendo detalhes sobre o erro ocorrido.

Exemplo de Uso: Facilita a depuração de problemas durante o processo de carregamento de dados de reuniões.

2.4 Regras de Negócio

No âmbito deste projeto, a implementação de regras de negócio visa garantir a integridade e consistência dos dados manipulados pelo sistema. Cada classe de regra de negócio possui métodos estáticos responsáveis por validar objetos específicos, assegurando que atendam aos requisitos necessários para o correto funcionamento do sistema.

Figura 13: Diagrama das Regras de Negócio

```
1 referência
public static class CondominioRegras
{
    0 referências
    public static void ValidarCondominio(Condominio condominio)
    {
        if (string.IsNullOrWhiteSpace(condominio.Nome))
        {
            throw new CondominioException.NomeCondominioNuloOuVazioException();
        }

        if (string.IsNullOrWhiteSpace(condominio.Endereco))
        {
            throw new CondominioException.EnderecoCondominioNuloOuVazioException();
        }

        if (condominio.Despesas.Count == 0)
        {
            throw new CondominioException.DespesasCondominioVaziasException();
        }

        if (condominio.Imoveis.Count == 0)
        {
            throw new CondominioException.ImoveisCondominioVaziosException();
        }

        if (condominio.Proprietarios.Count == 0)
        {
            throw new CondominioException.ProprietariosCondominioVaziosException();
        }

        if (condominio.Reunioes.Count == 0)
        {
            throw new CondominioException.ReunioesCondominioVaziasException();
        }

        if (condominio.Documentos.Count == 0)
        {
            throw new CondominioException.DocumentosCondominioVaziosException();
        }
    }
}
```

Figura 14: Regras de Negócio (CondominioRegras)

2.4.1 CondominioRegras

A classe "*CondominioRegras"* contém um método estático chamado "*ValidarCondominio*", cujo objetivo é validar se um objeto do tipo "*Condominio*" atende a determinadas regras de negócio. Abaixo estão as regras e suas condições correspondentes:

"*string.IsNullOrWhiteSpace(condominio.Nome)*":

Regra: O nome do condomínio não pode ser nulo ou vazio.

Ação: Lança uma exceção "*NomeCondominioNuloOuVazioException*" se o nome do condomínio for nulo ou vazio.

"*string.IsNullOrWhiteSpace(condominio.Endereco)*":

Regra: O endereço do condomínio não pode ser nulo ou vazio.

Ação: Lança uma exceção "*EnderecoCondominioNuloOuVazioException*" se o endereço do condomínio for nulo ou vazio.

"*condominio.Despesas.Count == 0*":

Regra: A lista de despesas do condomínio não pode estar vazia.

Ação: Lança uma exceção "*DespesasCondominioVaziasException*" se a lista de despesas do condomínio estiver vazia.

"*condominio.Imoveis.Count == 0*":

Regra: A lista de imóveis do condomínio não pode estar vazia.

Ação: Lança uma exceção "*ImoveisCondominioVaziosException*" se a lista de imóveis do condomínio estiver vazia.

"*condominio.Proprietarios.Count == 0*"

Regra: A lista de proprietários do condomínio não pode estar vazia.

Ação: Lança uma exceção "*ProprietariosCondominioVaziosException*" se a lista de proprietários do condomínio estiver vazia.

"*condominio.Reunioes.Count == 0*"

Regra: A lista de reuniões do condomínio não pode estar vazia.

Ação: Lança uma exceção "*ReunioesCondominioVaziasException*" se a lista de reuniões do condomínio estiver vazia.

"*condominio.Documentos.Count == 0*"

Regra: A lista de documentos do condomínio não pode estar vazia.

Ação: Lança uma exceção "*DocumentosCondominioVaziosException*" se a lista de documentos do condomínio estiver vazia.

2.4.2 DespesaRegras

A classe "*DespesaRegras*" contém um método estático chamado "*ValidarDespesa*", que recebe um objeto do tipo "*Despesa*" como parâmetro. Abaixo estão as regras e suas respetivas condições:

“*string.IsNullOrWhiteSpace(despesa.Tipo)*”:

Regra: O tipo da despesa não pode ser nulo ou vazio.

Ação: Lança uma exceção “*TipoDespesaNuloOuVazioException*” se o tipo da despesa for nulo ou vazio.

“*string.IsNullOrWhiteSpace(despesa.Imovel)*”:

Regra: O campo "*Imovel*" associado à despesa não pode ser nulo ou vazio.

Ação: Lança uma exceção “*ImovelDespesaNuloOuVazioException*” se o campo "Imovel" for nulo ou vazio.

“*despesa.Valor <= 0*”:

Regra: O valor da despesa deve ser maior que zero.

Ação: Lança uma exceção “*ValorDespesaInvalidoException*” se o valor da despesa for menor ou igual a zero.

“*despesa.DataVencimento < DateTime.Today*”:

Regra: A data de vencimento da despesa não pode ser uma data passada.

Ação: Lança uma exceção “*DataVencimentoDespesaPassadaException*” se a data de vencimento da despesa for anterior à data atual.

2.4.3 DocumentoRegras

A classe “*DocumentoRegras*” contém um método estático chamado “*ValidarDocumento*”, cujo propósito é validar se um objeto do tipo “*Documento*” atende a determinadas regras de negócio. Abaixo estão as regras e suas condições correspondentes:

“*string.IsNullOrWhiteSpace(documento.Tipo)*”:

Regra: O tipo do documento não pode ser nulo ou vazio.

Ação: Lança uma exceção “*TipoDocumentoNuloOuVazioException*” se o tipo do documento for nulo ou vazio.

"*string.IsNullOrWhiteSpace(documento.Nome)*":

Regra: O nome do documento não pode ser nulo ou vazio.

Ação: Lança uma exceção "*NomeDocumentoNuloOuVazioException*" se o nome do documento for nulo ou vazio.

"*string.IsNullOrWhiteSpace(documento.Conteudo)*":

Regra: O conteúdo do documento não pode ser nulo ou vazio.

Ação: Lança uma exceção "*ConteudoDocumentoNuloOuVazioException*" se o conteúdo do documento for nulo ou vazio.

2.4.4 ImovelRegras

A classe "*ImovelRegras*" contém um método estático chamado "*ValidarImovel*", cujo propósito é validar se um objeto do tipo "*Imovel*" atende a determinadas regras de negócio. Abaixo estão as regras e suas respetivas condições:

"*string.IsNullOrWhiteSpace(imovel.Endereco)*":

Regra: O endereço do imóvel não pode ser nulo ou vazio.

Ação: Lança uma exceção "*EnderecoImovelNuloOuVazioException*" se o endereço do imóvel for nulo ou vazio.

"*imovel.Despesas.Count == 0*":

Regra: O imóvel deve ter pelo menos uma despesa associada.

Ação: Lança uma exceção "*DespesasImovelVaziasException*" se a lista de despesas associadas ao imóvel estiver vazia.

"*imovel.Proprietarios.Count == 0*":

Regra: O imóvel deve ter pelo menos um proprietário associado.

Ação: Lança uma exceção "*ProprietariosImovelVaziosException*" se a lista de proprietários associados ao imóvel estiver vazia.

2.4.5 ProprietarioRegras

A classe "*ProprietarioRegras*" possui um método estático chamado "*ValidarProprietario*" que tem como objetivo validar se um objeto do tipo "*Proprietario*" atende a determinadas regras de negócio. Abaixo estão as regras e suas condições correspondentes:

"*string.IsNullOrWhiteSpace(proprietario.Nome)*":

Regra: O nome do proprietário não pode ser nulo ou vazio.

Ação: Lança uma exceção "*NomeProprietarioNuloOuVazioException*" se o nome do proprietário for nulo ou vazio.

"*string.IsNullOrWhiteSpace(proprietario.Contato)*":

Regra: O contato do proprietário não pode ser nulo ou vazio.

Ação: Lança uma exceção "*ContatoProprietarioNuloOuVazioException*" se o contato do proprietário for nulo ou vazio.

"*string.IsNullOrWhiteSpace(proprietario.Imovel)*":

Regra: O proprietário deve estar associado a um imóvel.

Ação: Lança uma exceção "*ImovelProprietarioNuloOuVazioException*" se o imóvel associado ao proprietário for nulo ou vazio.

"*string.IsNullOrWhiteSpace(proprietario.Nif)*":

Regra: O NIF (Número de Identificação Fiscal) do proprietário não pode ser nulo ou vazio.

Ação: Lança uma exceção "*NifProprietarioNuloOuVazioException*" se o NIF do proprietário for nulo ou vazio.

2.4.6 ReuniaoRegras

A classe "*ReuniaoRegras*" possui um método estático chamado "*ValidarReuniao*", que tem como objetivo validar se um objeto do tipo "*Reuniao*" atende a determinadas regras de negócio. Abaixo estão as regras e suas condições correspondentes:

"*reuniao.Data < DateTime.Now*":

Regra: A data da reunião deve ser igual ou posterior à data atual.

Ação: Lança uma exceção "*DataReuniaoInvalidaException*" se a data da reunião for anterior à data atual.

"*reuniao.Hora < TimeSpan.Zero || reuniao.Hora >= TimeSpan.FromDays(1)*":

Regra: A hora da reunião deve estar no intervalo entre zero e 24 horas.

Ação: Lança uma exceção "*HoraReuniaoInvalidaException*" se a hora da reunião for menor que zero ou maior ou igual a 24 horas.

"*string.IsNullOrWhiteSpace(reuniao.Local)*":

Regra: O local da reunião não pode ser nulo ou vazio.

Ação: Lança uma exceção "*LocalReuniaoNuloOuVazioException*" se o local da reunião for nulo ou vazio.

"*reuniao.Intervenientes.Count == 0*":

Regra: Deve haver pelo menos um interveniente na reunião.

Ação: Lança uma exceção "*IntervenientesReuniaoVaziosException*" se a lista de intervenientes na reunião estiver vazia.

2.5 Interfaces

As interfaces desempenham um papel crucial no desenvolvimento de software, estabelecendo uma espécie de contratos que as classes devem seguir. No âmbito deste sistema de gestão de condomínios, diversas interfaces foram criadas para definir requisitos

mínimos que as classes associadas aos elementos principais do sistema devem cumprir. Cada interface representa um conjunto específico de propriedades que as classes correspondentes devem implementar, garantindo assim uma estrutura consistente e facilitando a manutenção e expansão do código.

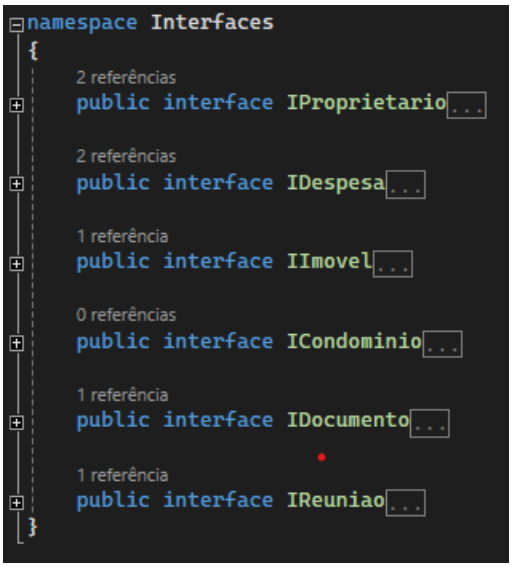

Figura 15: Interfaces

2.5.1 ICondominio

A interface "*ICondominio*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Aqui está uma explicação detalhada de cada membro da interface:

Propriedade "*Nome*":

Tipo: "*string*".

Descrição: Obtém ou define o nome do condomínio.

Propriedade "*Endereco*":

Tipo: "*string*".

Descrição: Obtém ou define o endereço do condomínio.

Propriedade "*Despesas*":

Tipo: "*List<IDespesa>*".

Descrição: Obtém ou define a lista de despesas associadas ao condomínio.

Propriedade "*Imoveis*":

Tipo: "*List<IImovel>*".

Descrição: Obtém ou define a lista de imóveis associados ao condomínio.

Propriedade "*Proprietarios*":

Tipo: "*List<IProprietario>*".

Descrição: Obtém ou define a lista de proprietários associados ao condomínio.

Propriedade "*Reunioes*":

Tipo: "*List<IReuniao>*".

Descrição: Obtém ou define a lista de reuniões agendadas no condomínio.

Propriedade "*Documentos*":

Tipo: "*List<IDocumento>*".

Descrição: Obtém ou define a lista de documentos associados ao condomínio.

A interface "*ICondominio*" define uma espécie de contrato que as classes que representam condomínios devem seguir, garantindo consistência na implementação.

2.5.2 IDespesa

A interface "*IDespesa*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Aqui está uma explicação detalhada de cada membro da interface:

Propriedade "*Tipo*":

Tipo: "*string*".

Descrição: Obtém ou define o tipo da despesa.

Propriedade "*Valor*":

Tipo: "*decimal*".

Descrição: Obtém ou define o valor da despesa.

Propriedade "*DataVencimento*":

Tipo: "*DateTime*".

Descrição: Obtém ou define a data de vencimento da despesa.

Propriedade "*EstadoPagamento*":

Tipo: "*bool*".

Descrição: Obtém ou define o estado de pagamento da despesa.

Propriedade "*Imovel*":

Tipo: "*string*".

Descrição: Obtém ou define o identificador único do imóvel associado à despesa.

A interface "*IDespesa*" é uma espécie de contrato que define um conjunto mínimo de propriedades necessárias para representar uma despesa no sistema.

2.5.3 IDocumento

A interface "*IDocumento*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Aqui está uma explicação detalhada de cada membro da interface:

Propriedade "*Nome*":

Tipo: "*string*".

Descrição: Obtém ou define o nome do documento.

Propriedade "*Tipo*":

Tipo: "*string*".

Descrição: Obtém ou define o tipo do documento.

Propriedade "*Conteudo*":

Tipo: "*string*".

Descrição: Obtém ou define o conteúdo do documento.

A interface "*IDocumento*" é uma espécie de contrato que define um conjunto mínimo de propriedades necessárias para representar um documento no sistema.

2.5.4 IImovel

A interface "*IImovel*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Aqui está uma explicação detalhada de cada membro da interface:

Propriedade "*IdImovel*":

Tipo: "*int*".

Descrição: Obtém ou define o identificador único do imóvel.

Propriedade "*Proprietarios*":

Tipo: "*List<IProprietario>*".

Descrição: Obtém ou define a lista de proprietários associados ao imóvel.

Propriedade "*Despesas*":

Tipo: "*List<IDespesa>*".

Descrição: Obtém ou define a lista de despesas associadas ao imóvel.

Propriedade "*Quotas*":

Tipo: "*List<decimal>*".

Descrição: Obtém ou define a lista de quotas associadas ao imóvel.

Propriedade "*Endereco*":

Tipo: "*string*".

Descrição: Obtém ou define o endereço associado ao imóvel.

A interface "*IImovel*" é uma espécie de contrato que define um conjunto mínimo de propriedades necessárias para representar um imóvel no sistema.

2.5.5 IProprietario

A interface "*IProprietario*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Aqui está uma explicação detalhada de cada membro da interface:

Propriedade "*Nome*":
Tipo: "*string*".
Descrição: Obtém ou define o nome do proprietário.

Propriedade "*Contato*":
Tipo: "*string*".
Descrição: Obtém ou define o número de telefone do proprietário.

Propriedade "*Imovel*":
Tipo: "*string*".
Descrição: Obtém ou define o identificador único do imóvel do proprietário.

Propriedade "*Nif*":
Tipo: "*string*".
Descrição: Obtém ou define o número de identificação fiscal do proprietário.

A interface "*IProprietario*" é uma espécie de contrato que define um conjunto mínimo de propriedades necessárias para representar um proprietário no sistema.

2.5.6 IReuniao

A interface "*IReuniao*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Aqui está uma explicação detalhada de cada membro da interface:

Propriedade "*Data*":
Tipo: "*DateTime*".
Descrição: Obtém ou define a data da reunião.

Propriedade "*Hora*":
Tipo: "*TimeSpan*".
Descrição: Obtém ou define a hora da reunião.

Propriedade "*Local*":
Tipo: "*string*".
Descrição: Obtém ou define o local da reunião.

Propriedade "*Intervenientes*":
Tipo: "*List<string>*".
Descrição: Obtém ou define a lista de intervenientes na reunião.

A interface "*IReuniao*" é uma espécie de contrato que define um conjunto mínimo de propriedades necessárias para representar uma reunião no sistema.

2.6 Gestor (Program)

[8], [9]O programa segue uma abordagem orientada a objetos, utilizando diversas classes para modelar entidades relacionadas a condomínios, proprietários, imóveis, despesas, reuniões e documentos. Cada classe está contida em seu próprio arquivo e está organizada em "*namespaces*" correspondentes.

Classes Principais:

- **"*Program*":** Esta é a classe principal que contém o método "*Main*". Neste método, são criadas instâncias de objetos fictícios, representando um condomínio, um proprietário, um imóvel, uma despesa, uma reunião e um documento. Em seguida, são invocados métodos de validação específicos de cada objeto por meio do método "*ValidarEExecutar*".

Objetos de Negócio:

- Cada classe de objeto de negócio ("*Condominio*", "*Proprietario*", "*Imovel*", "*Despesa*", "*Reuniao*" e "*Documento*") possui propriedades e métodos específicos para representar suas características e comportamentos. O uso de classes favorece a encapsulação e a modularidade.

Regras de Negócio:

- **"*CondominioRegras*":** Contém regras específicas de validação para objetos do tipo `Condominio`.
- **"*DespesaRegras*":** Define regras de validação para objetos do tipo "*Despesa*".
- **"*ImovelRegras*":** Contém regras de validação para objetos do tipo "*Imovel*".
- **"*ProprietarioRegras*":** Define regras específicas de validação para objetos do tipo "*Proprietario*".
- **"*ReuniaoRegras*":** Contém regras específicas de validação para objetos do tipo "*Reuniao*".
- **"*DocumentoRegras*":** Define regras de validação para objetos do tipo "*Documento*".

Exceções:

- Cada regra de negócio possui exceções específicas (por exemplo, "*CondominioException*", "*DespesaException*") que herdam de uma classe base "*Exception*". Essas exceções personalizadas permitem um tratamento mais preciso de erros relacionados às validações.

Método "*ValidarEExecutar*":

- Este método é responsável por validar e executar ações específicas, tratando exceções de negócio de forma adequada. Ele aceita um "*Action*" como parâmetro para representar a ação a ser executada.

Fluxo de Execução:

- O programa inicia com a criação de instâncias fictícias de objetos, seguida pela validação de cada objeto por meio do método "*ValidarEExecutar*".
- As exceções específicas são tratadas e mensagens detalhadas são exibidas no console, indicando o tipo de erro e a origem.

```
// Criar reuniao usando os dados fornecidos
var reuniao = new Reuniao
{
    Data = DateTime.Now.AddDays(7),
    Hora = new TimeSpan(14, 30, 0),
    Local = "Sala de Condomínio",
    Intervenientes = new System.Collections.Generic.List<string> { "Membro1", "Membro2" }
};
```

Figura 16: Program (Reunião fictícia)

```
try
{
    // Validar e executar as regras de negócio para cada objeto
    ValidarEExecutar(() => CondominioRegras.ValidarCondominio(condominio));
    ValidarEExecutar(() => DespesaRegras.ValidarDespesa(despesa));
    ValidarEExecutar(() => ImovelRegras.ValidarImovel(imovel));
    ValidarEExecutar(() => ProprietarioRegras.ValidarProprietario(proprietario));
    ValidarEExecutar(() => ReuniaoRegras.ValidarReuniao(reuniao));
    ValidarEExecutar(() => DocumentoRegras.ValidarDocumento(documento));
}
catch (Exception ex)
{
    // Tratar exceção genérica, se necessário
    Console.WriteLine($"Erro: {ex.Message}");
}
```

Figura 17: Program (Validação de Objetos)

```
static void ValidarEExecutar(Action acao)
{
    try
    {
        acao.Invoke();
    }
    catch (CondominioException ex)
    {
        Console.WriteLine($"Erro de Condominio: {ex.Message}");
    }
    catch (DespesaException ex)
```

Figura 18: Program (Método de validar e execução)

3. Conclusão

Em conclusão, o desenvolvimento deste programa de gestão de condomínios representou uma aplicação prática dos princípios fundamentais da Programação Orientada a Objetos (POO). Através de uma abordagem estruturada e modular, procurou-se simplificar e otimizar os desafios inerentes à gestão de condomínios. Desde a representação de propriedades, proprietários, despesas e reuniões até a implementação de regras de negócio específicas, cada componente do sistema reflete um compromisso com a eficiência e a organização.

A hierarquia de classes foi cuidadosamente projetada para refletir a estrutura do mundo real, garantindo uma representação fiel dos objetos envolvidos na gestão de condomínios. A utilização de exceções personalizadas, regras de negócio e a integração de validações específicas contribuem para a robustez e confiabilidade do sistema.

Ao longo deste relatório, explorou-se detalhadamente cada componente, examinando as decisões de design que fundamentam a arquitetura do sistema.

4. Referências

[1] "Resolver avisos anuláveis - C# | Microsoft Learn." Accessed: Dec. 20, 2023. [Online]. Available: https://learn.microsoft.com/pt-pt/dotnet/csharp/language-reference/compiler-messages/nullable-warnings

[2] "Doxygen: Doxygen." Accessed: Dec. 20, 2023. [Online]. Available: https://www.doxygen.nl/

[3] "Design, visualize, & refactor with Class Designer - Visual Studio (Windows) | Microsoft Learn." Accessed: Dec. 20, 2023. [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/ide/class-designer/designing-and-viewing-classes-and-types?view=vs-2022

[4] "Doxygen Class Diagram and Document Auto generation from Code, #ClassDiagram #UML #Documentation - YouTube." Accessed: Dec. 20, 2023. [Online]. Available: https://www.youtube.com/watch?v=oamo349N34Y&ab_channel=AMR.SMART.SYSTEMS

[5] E. Filipe and G. Alves, "Guia Doxygen", Accessed: Dec. 20, 2023. [Online]. Available: http://www.stack.nl/~dimitri/doxygen/manual/index.html

[6] "Serialização e desserialização em C# com exemplo." Accessed: Dec. 20, 2023. [Online]. Available: https://www.guru99.com/pt/c-sharp-serialization.html

[7] "Como serializar JSON em C# - .NET | Microsoft Learn." Accessed: Dec. 20, 2023. [Online]. Available: https://learn.microsoft.com/pt-pt/dotnet/standard/serialization/system-text-json/how-to

[8] "Compiler Error CS1729 - C# | Microsoft Learn." Accessed: Dec. 20, 2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/compiler-messages/cs1729?f1url=%3FappId%3Droslyn%26k%3Dk(CS1729)

[9] "Fixing Compiler Error CS7036 In C# Application - YouTube." Accessed: Dec. 20, 2023. [Online]. Available: https://www.youtube.com/watch?app=desktop&v=429vpkZ4ysk&ab_channel=BANSODETECHSOLUTION