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

Índice de Acrónimos

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. Idealização do sistema de Gestão de Condomínios

O sistema de gestão de condomínios apresenta uma estrutura robusta e abrangente, centrada em classes que contém informações vitais para a administração eficiente e colaborativa de condomínios. A classe central, "Gestão", assume um papel importante ao coordenar e executar operações necessárias para o bom funcionamento do condomínio. Esta classe serve como um ponto central para a execução de processos importantes e para a manutenção do fluxo de informações entre as diversas classes do sistema. Pela figura 1, percebe-se que existe mais classes além da classe "Gestão", sendo estas: "Condomínio", "Imóvel", "Proprietário", "Despesa", "Reuniao" e "Documento".

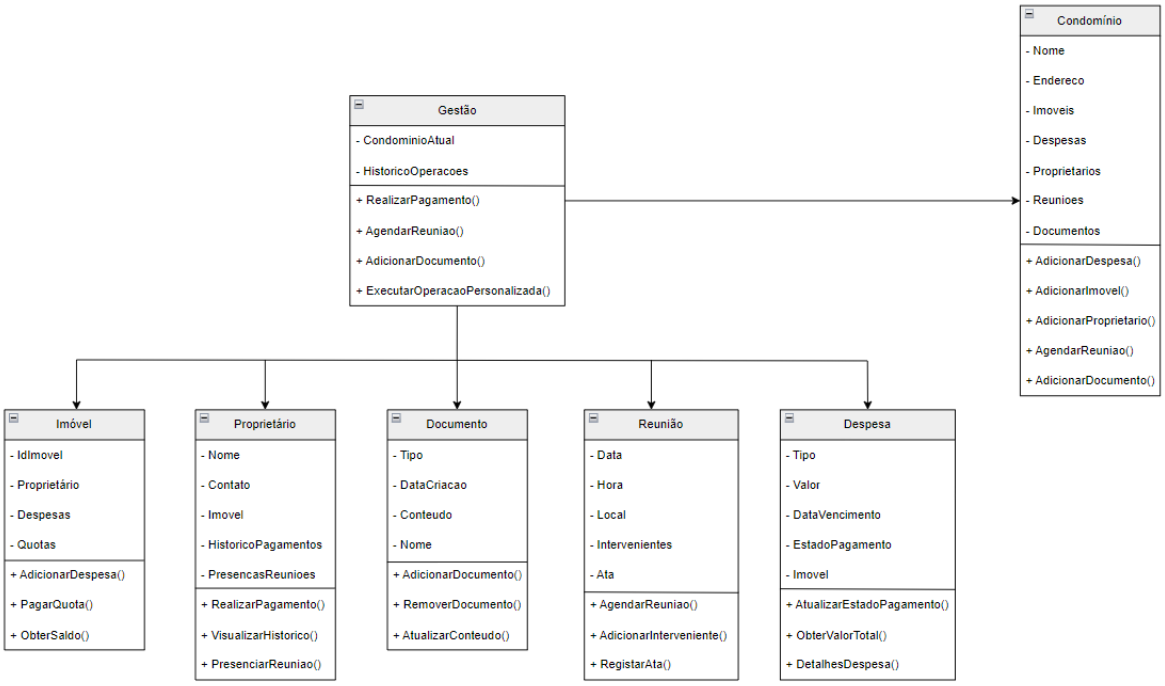

Figura 1: Diagrama de Classes

Com base no diagrama de classes (figura 1), é possível observar as heranças entre as classes. Assim, no contexto das heranças, temos:

- **"Gestão" → "Condomínio":** A classe "Condomínio" herda características e operações da classe "Gestão". Isso significa que a gestão do condomínio é realizada através da classe "Gestão", que possui métodos e atributos essenciais,

e a classe "Condomínio" estende esses recursos para lidar com as operações específicas de condomínio.

- **"Gestão" → "Imóvel", "Proprietário", "Despesa", "Reunião" e "Documento":** Cada uma delas herda diretamente da classe "Gestão". Isso significa que cada classe no sistema (Imóvel, Proprietário, Despesa, Reunião e Documento) tem acesso direto aos métodos e atributos da classe "Gestão", facilitando a coordenação centralizada dessas entidades.

2.1 Gestão

A classe "*Gestão*" desempenha um papel central no sistema de gestão de condomínios, sendo esta responsável por coordenar e executar operações necessárias para que haja um bom funcionamento do condomínio. Ela atua como o ponto central para a execução de processos importantes e para a manutenção do fluxo de informações entre as diversas entidades do sistema.

Atributos:

1. **"*CondominioAtual*":** Referência ao objeto "*Condominio*" que está sendo gerenciado pelo gestor;
2. **"*HistoricoOperacoes*":** Registo das operações executadas pelo gestor ao longo do tempo.

Métodos:

1. **"*RealizarPagamento()*":** Iniciar o processo de registo de pagamento para um proprietário específico;
2. **"*AgendarReuniao()*":** Agendar uma nova reunião no condomínio;
3. **"*AdicionarDocumento()*":** Adicionar um novo documento ao sistema;
4. **"*ExecutarOperacaoOPersonalizada()*":** Permite executar operações personalizadas no contexto do condomínio.

2.2 Condomínio

A classe "*Condomínio*" representa a peça fundamental no sistema de gestão de condomínios, contendo todas as informações relevantes sobre um condomínio específico. Seu papel é essencial na organização e administração dos diversos aspetos que compõem a gestão de condomínios. Esta classe serve como uma classe central, concentrando dados importantes que são essenciais para o bom funcionamento e colaborativo do condomínio.

Atributos:

1. **"*Nome*":** Representa o nome do condomínio;
2. **"*Endereco*":** Endereço físico do condomínio;
3. **"*Imoveis*":** Contém uma lista de imóveis existentes no condomínio;
4. **"*Despesas*":** Contém uma lista de despesas associadas ao condomínio (água, eletricidade, limpeza, manutenção…);
5. **"*Proprietarios*":** Contém uma lista de proprietários que fazem parte do condomínio;
6. **"*Reunioes*":** Contém informações sobre reuniões agendadas (data, hora, local e pauta);
7. **"*Documentos*":** Contém documentos relevantes para o condomínio (atas de reunião, regulamentos internos, comunicados…).

Métodos:

1. **"*AdicionarDespesa()*":** Permite adicionar uma nova despesa à lista de despesas do condomínio;
2. **"*AdicionarImovel()*":** Adicionar um novo imóvel à lista de imoveis do condomínio;
3. **"*AdicionarProprietario()*":** Adicionar um novo proprietário à lista de proprietários do condomínio;

4. **"*AgendarReuniao()*":** Agendar uma nova reunião para o condomínio;
5. **"*AdicionarDocumento()*":** Adicionar um novo documento à lista de documentos do condomínio.

2.3 Imóvel

A classe "*Imóvel*" é uma representação fundamental no sistema de gestão de condomínios, conter informações específicas sobre um imóvel dentro do condomínio. Esta classe é essencial para seguir as responsabilidades financeiras e associativas de cada imóvel.

Atributos:

1. **"*IdImovel*":** Identificação do imóvel dentro do condomínio através do número ou nome;
2. **"*Proprietario*":** Identificação do proprietário do imóvel em questão;
3. **"*Despesas*":** Lista das despesas associadas ao imóvel;
4. **"*Quotas*":** Registo das quotas pagas.

Métodos:

1. **"*AdicionarDespesa()*":** Adicionar uma nova despesa ao imóvel;
2. **"*PagarQuota()*":** Registar o pagamento de uma quota, atualizando o histórico de quotas pagas;
3. **"*ObterSaldo()*":** Calcular e retornar o saldo atual da unidade, tendo atenção às despesas e quotas pagas.

2.4 Proprietário

A classe "*Proprietário*" representa as pessoas que possuem imóveis do condomínio, sendo essenciais para a gestão participativa e colaborativa. Esta classe contém informações sobre os proprietários ou inquilinos dos imóveis, facilitando o acompanhamento de dados pessoais e contribuições para o condomínio.

Atributos:

1. **"*Nome*":** O nome do proprietário;
2. **"*Contato*":** O número de telemóvel e email;
3. **"*Imovel*":** Referência ao imóvel à qual o proprietário está associado;
4. **"*HistoricoPagamentos*":** Lista das contribuições financeiras do proprietário ao longo do tempo;
5. **"*PresencasReunioes*":** Lista das presenças do proprietário em reuniões do condomínio.

Métodos:

1. **"*RealizarPagamento()*":** Registar o pagamento do proprietário, atualizando o histórico de pagamentos;
2. **"*VisualizarHistorico()*":** Mostrar todo o histórico de pagamentos realizados pelo proprietário;
3. **"*ParticiparReuniao()*":** Registar a presença do proprietário em uma reunião especifica.

2.5 Despesa

A classe "*Despesa*" é fundamental no sistema de gestão de condomínios, representando cada uma das obrigações financeiras que o condomínio deve suportar.

Esta classe contém informações detalhadas sobre as despesas, ajudando assim na gestão e na distribuição justa dos custos entre os imóveis.

Atributos:

1. **"*Tipo*":** Indica o tipo de despesa (água, eletricidade, limpeza, manutenção…)
2. **"*Valor*":** Indica o montante financeiro associado à despesa;
3. **"*DataVencimento*":** Indica a data-limite para realizar o pagamento da despesa;
4. **"*EstadoPagamento*":** Regista se a despesa foi paga ou está pendente;
5. **"*Imovel*":** Referência ao imóvel à qual a despesa está associada.

Métodos:

1. **"*AtualizarEstadoPagamento()*":** Atualizar o estado de pagamento da despesa;
2. **"*ObterValorTotal()*":** Mostrar o valor total da despesa;
3. **"*DetalhesDespesa()*":** Mostrar detalhadamente a despesa.

2.6 Reunião

A classe "*Reunião*" desempenha um papel central no sistema de gestão de condomínios, proporcionando um meio estruturado para discussões, decisões e comunicação entre os proprietários dos imoveis. Esta classe contém informações essenciais sobre as reuniões realizadas no âmbito do condomínio.

Atributos:

1. **"*Data*":** Indica a data em que a reunião esta agendada para ocorrer;
2. **"*Hora*":** Indica a hora em que a reunião esta marcada para começar;

3. **"*Local*":** Indica o local onde a reunião será realizada;
4. **"*Intervenientes*":** Lista dos proprietários que vão participar na reunião;
5. **"*Ata*":** Documento que regista as decisões e discussões ocorridas durante a reunião.

Métodos:

1. **"*AgendarReuniao()*":** Agendar uma nova reunião com os detalhes fornecidos;
2. **"*AdicionarInterveniente()*":** Adicionar um proprietário à lista de intervenientes da reunião;
3. **"*RegistarAta()*":** Associar um documento do tipo ata à reunião, registando as decisões e discussões.

2.7 Documento

A classe "*Documento*" é uma classe crucial no sistema de gestão de condomínios, visto que este representa os registos formais e informações essenciais. Esta classe contém diversos tipos de documentos, como atas de reuniões, regulamentos internos, comunicados e outros, fornecendo uma estrutura padrão para o armazenamento e procuração de informações importantes.

Atributos:

1. **"*Nome*":** O nome do documento;
2. **"*Tipo*":** Indica o tipo de documento (ata, regulamento, comunicado…);
3. **"*DataCriacao*":** A data em que o documento foi criado ou registado;
4. **"*Conteudo*":** O conteúdo informativo do documento.

Métodos:

1. **"*AdicionarDocumento()*":** Adicionar um novo documento ao sistema;

2. **"*RemoverDocumento()*":** Remover um documento existente com base no nome fornecido;
3. **"*AtualizarConteudo()*":** Atualizar o conteúdo de um documento específico.

3. Implementação do sistema de Gestão de Condomínios

3.1 Objetos de Negócio

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.

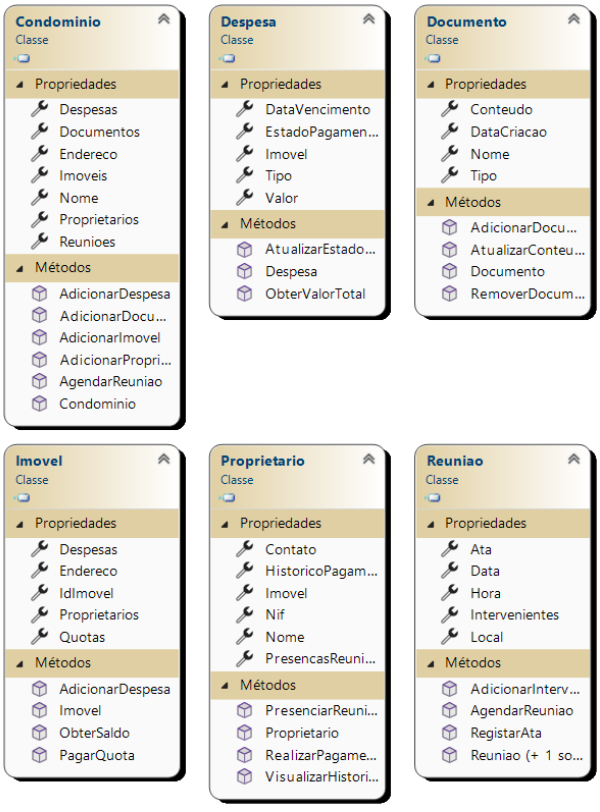

Figura 2: Diagrama de Classes (Objetos de Negócio)

3.1.1 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.
- **"*endereço*":** 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.

```
#region Atributos

// Nome do condomínio
private string nome = null!;

// Endereço do condomínio
private string endereco = null!;

// Lista de despesas associadas ao condomínio
private List<string> despesas = null!;

// Lista de imóveis associados ao condomínio
private List<string> imoveis = null!;

// Lista de proprietários associados ao condomínio
private List<string> proprietarios = null!;

// Lista de reuniões agendadas no condomínio
private List<string> reunioes = null!;

// Lista de documentos associados ao condomínio
private List<string> documentos = null!;

#endregion
```

Figura 3: Objetos de Negócio (Condominio - Atributos)

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 4: 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 marcada como "*Serializable*", indicando que os objetos dela podem ser serializados para fins de armazenamento ou transmissão.

Durante a inicialização dos atributos, é utilizado "*null!*" para indicar que esses atributos não são permitidos serem nulos. O uso do operador "*!*" (assertividade) foi necessário para garantir que os atributos estejam sempre inicializados antes de serem usados, contribuindo para a funcionalidade correta do código. Certifique-se de que as listas sejam devidamente inicializadas antes de serem utilizadas para evitar exceções em tempo de execução.

Não há implementação de métodos "*Override*" ou "*Destrutor*" nesta classe.

3.1.2 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.

- **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 marcada como "*Serializable*", indicando que os objetos dela podem ser serializados para fins de armazenamento ou transmissão.

Durante a inicialização, os atributos são definidos com "*null!*", indicando que esses atributos não são permitidos serem nulos. O uso do operador "*!*" (assertividade) é necessário para garantir que os atributos estejam sempre inicializados antes de serem usados, contribuindo para a funcionalidade correta do código.

Não há implementação de métodos "*Override*" ou "*Destrutor*" nesta classe.

3.1.3 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.
- **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*", indicando que os objetos dela podem ser serializados para fins de armazenamento ou transmissão.

Durante a inicialização, os atributos são definidos com "*string.Empty*" e "*DateTime.MinValue*", garantindo que os atributos estejam sempre inicializados antes de serem usados, contribuindo para a funcionalidade correta do código.

Não há implementação de métodos "*Override*" ou "*Destrutor*" nesta classe.

3.1.4 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.

- **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 marcada como "*Serializable*", indicando que os objetos dela podem ser serializados para fins de armazenamento ou transmissão.

Durante a inicialização, as listas de despesas e quotas são inicializadas para evitar exceções em tempo de execução.

Não há implementação de métodos "*Override*" ou "*Destrutor*" nesta classe.

3.1.5 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.

- **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 marcada como "*Serializable*", indicando que os objetos dela podem ser serializados para fins de armazenamento ou transmissão.

Durante a inicialização, as listas de histórico de pagamentos e presenças em reuniões são inicializadas para evitar exceções em tempo de execução.

Não há implementação de métodos "*Override*" ou "*Destrutor*" nesta classe.

3.1.6 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.

- **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*", indicando que os objetos dela podem ser serializados para fins de armazenamento ou transmissão.

Durante a inicialização, a lista de intervenientes é inicializada para evitar exceções em tempo de execução.

Não há implementação de métodos "*Override*" ou "*Destrutor*" nesta classe.

3.2 Dados

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.

3.2.1 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 (Linhas 1-7)**

- **"*System*":** Fornece funcionalidades básicas do "*C#*".
- **"*System.Collections.Generic*":** Oferece interfaces e classes genéricas, como "*List<T>*".
- **"*ObjetosNegocio*":** "*Namespace*" que pode conter objetos de negócio relacionados ao código.
- **"*Excecoes*":** Namespace que contém classes de exceção personalizadas.
- **"*RegrasNegocio*":** Namespace que contém regras de negócio específicas.
- **"*System.IO*":** Fornece classes para entrada e saída, como operações de arquivo.
- **"*System.Text.Json*":** Oferece funcionalidades de serialização e desserialização "*JSON*".

2. **Declaração do "*Namespace*" e da Classe (Linhas 9-13)**

- **"*namespace Dados*":** Define o "*namespace*" onde a classe está contida.
- **"*public class Condominios*":** Declaração da classe "*Condominios*" responsável por armazenar e gerenciar informações sobre condomínios.

3. **Atributos e Regiões (Linhas 15-22)**

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

4. **Construtores (Linhas 24-30)**

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

5. **Métodos Principais (Linhas 32-106)**

- **"*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.

- **"*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.

A classe *Condominios*" segue a mesma estrutura organizada e coerente das classes anteriores. Ela fornece métodos bem definidos para adição, procura, gravação e carregamento de informações sobre condomínios, contribuindo para uma gestão eficiente desses dados no contexto do sistema.

3.2.2 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 (Linhas 1-7)**

- **"*System*":** Fornece funcionalidades básicas do "*C#*".
- **"*System.Collections.Generic*":** Oferece interfaces e classes genéricas, como "*List<T>*".

- **"*ObjetosNegocio*":** "*Namespace*" que pode conter objetos de negócio relacionados ao código.
- **"*Excecoes*":** "*Namespace*" que contém classes de exceção personalizadas.
- **"*RegrasNegocio*":** "*Namespace*" que contém regras de negócio específicas.
- **"*System.IO*":** Fornece classes para entrada e saída, como operações de arquivo.
- **"*System.Text.Json*":** Oferece funcionalidades de serialização e desserialização "*JSON*".

2. Declaração do "*Namespace*" e da Classe (Linhas 9-13)

- **"*namespace Dados*":** Define o "*namespace*" onde a classe está contida.
- **"*public class Despesas*":** Declaração da classe "*Despesas*" responsável por armazenar e gerenciar informações sobre despesas.

3. Atributos e Regiões (Linhas 15-22)

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

4. Construtores (Linhas 24-30)

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

5. Métodos Principais (Linhas 32-106)

- **"*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.

A classe "*Despesas*" segue a mesma estrutura organizada e coerente das classes anteriores. Ela fornece métodos bem definidos para adição, procura, gravação e carregamento de informações sobre despesas, contribuindo para uma gestão eficiente desses dados no contexto do sistema.

3.2.3 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 (Linhas 1-7)**

- **"System":** Fornece funcionalidades básicas do "*C#*".
- **"System.Collections.Generic":** Oferece interfaces e classes genéricas, como "*List<T>*".
- **"ObjetosNegocio":** "*Namespace*" que pode conter objetos de negócio relacionados ao código.
- **"Excecoes":** "*Namespace*" que contém classes de exceção personalizadas.
- **"RegrasNegocio":** "*Namespace*" que contém regras de negócio específicas.
- **"System.Text.Json":** Oferece funcionalidades de serialização e desserialização "*JSON*".

2. **Declaração do "*Namespace*" e da Classe (Linhas 9-13)**

- **"namespace Dados":** Define o "*namespace*" onde a classe está contida.
- **"public class Documentos":** Declaração da classe "*Documentos*" responsável por armazenar e gerenciar informações sobre documentos.

3. **Atributos e Regiões (Linhas 15-22)**

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

4. **Construtores (Linhas 24-30)**

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

5. **Métodos Principais (Linhas 32-106)**

- **"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.

A classe "*Documentos*" segue a mesma estrutura organizada e coerente das classes anteriores. Ela fornece métodos bem definidos para adição, procura, gravação e carregamento de informações sobre documentos, contribuindo para uma gestão eficiente desses dados no contexto do sistema.

3.2.4 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 (Linhas 1-7)**

- **"*System*":** Fornece funcionalidades básicas do "*C#*".
- **"*System.Collections.Generic*":** Oferece interfaces e classes genéricas, como "*List<T>*".
- **"*ObjetosNegocio*":** "*Namespace*" que pode conter objetos de negócio relacionados ao código.
- **"*Excecoes*":** "*Namespace*" que contém classes de exceção personalizadas.
- **"*RegrasNegocio*":** "*Namespace*" que contém regras de negócio específicas.
- **"*System.IO*":** Fornece classes para entrada e saída, como operações de arquivo.
- **"*System.Text.Json*":** Oferece funcionalidades de serialização e desserialização "*JSON*".

2. **Declaração do "*Namespace*" e da Classe (Linhas 9-13)**

- **"*Namespace Dados*":** Define o "*namespace*" onde a classe está contida.
- **"*public class Imoveis*":** Declaração da classe "*Imoveis*" responsável por armazenar e gerenciar informações sobre imóveis.

3. Atributos e Regiões (Linhas 15-22)

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

4. Construtores (Linhas 24-30)

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

5. Métodos Principais (Linhas 32-106)

- **"*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.

A classe "*Imoveis*" segue a mesma estrutura organizada e coerente das classes anteriores. Ela fornece métodos bem definidos para adição, procura, gravação e carregamento de informações sobre imóveis, contribuindo para uma gestão eficiente desses dados no contexto do sistema.

3.2.5 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 (Linhas 1-7)**

- **"*System*":** Fornece funcionalidades básicas do "*C#*".
- **"*System.Collections.Generic*":** Oferece interfaces e classes genéricas, como "*List<T>*".
- **"*ObjetosNegocio*":** "*Namespace*" que pode conter objetos de negócio relacionados ao código.
- **"*Excecoes*":** "*Namespace*" que contém classes de exceção personalizadas.
- **"*RegrasNegocio*":** "*Namespace*" que contém regras de negócio específicas.
- **"*System.IO*":** Fornece classes para entrada e saída, como operações de arquivo.
- **"*System.Text.Json*":** Oferece funcionalidades de serialização e desserialização "*JSON*".

2. **Declaração do "*Namespace*" e da Classe (Linhas 9-13)**

- **"*namespace Dados*":** Define o "*namespace*" onde a classe está contida.
- **"*public class Proprietarios*":** Declaração da classe "*Proprietarios*" responsável por armazenar e gerenciar informações sobre proprietários.

3. **Atributos e Regiões (Linhas 15-22)**

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

4. **Construtores (Linhas 24-30)**

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

5. Métodos Principais (Linhas 32-106)

- **"*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.

A classe "*Proprietarios*" segue a mesma estrutura organizada e coerente das classes anteriores. Ela fornece métodos bem definidos para adição, procura, gravação e carregamento de informações sobre proprietários, contribuindo para uma gestão eficiente desses dados no contexto do sistema.

3.2.6 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)**

- **"*System*":** Fornece funcionalidades básicas do "*C#*".
- **"*System.Collections.Generic*":** Oferece interfaces e classes genéricas, como "*List<T>*".
- **"*ObjetosNegocio*":** "*Namespace*" que pode conter objetos de negócio relacionados ao código.
- **"*Excecoes*":** "*Namespace*" que contém classes de exceção personalizadas.
- **"*RegrasNegocio*":** "*Namespace*" que contém regras de negócio específicas.
- **"*System.IO*":** Fornece classes para entrada e saída, como operações de arquivo.
- **"*System.Text.Json*":** Oferece funcionalidades de serialização e desserialização "*JSON*".

2. **Declaração do "*Namespace*" e da Classe (Linhas 9-13)**

- **"*namespace Dados*":** Define o "*namespace*" onde a classe está contida.
- **"*public class Reunioes*":** Declaração da classe "*Reunioes*" responsável por armazenar e gerenciar informações sobre reuniões.

3. **Atributos e Regiões (Linhas 15-22)**

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

4. **Construtores (Linhas 24-30)**

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

5. **Métodos Principais (Linhas 32-106)**

- **"*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.

A classe "Reunioes" segue a mesma estrutura organizada e coerente das classes anteriores. Ela fornece métodos bem definidos para adição, procura, gravação e carregamento de informações sobre reuniões, contribuindo para uma gestão eficiente desses dados no contexto do sistema.

3.3 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:* Erros relacionados a condomínios.
2. *DespesaException:* Exceções específicas de despesas.
3. *DocumentoException:* Tratamento de erros relacionados a documentos.
4. *ImovelException:* Erros relativos a imóveis.
5. *ProprietarioException:* Exceções específicas de proprietários.

6. *ReuniaoException:* Lidar com erros relacionados a reuniões.

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

3.3.1 CondominioException

O domínio "*CondominioException*" engloba exceções relacionadas ao contexto de condomínios no sistema. Cada subclasse específica aborda situações distintas que podem resultar em erros na manipulação de informações sobre condomínios. Aqui estão as explicações detalhadas para cada exceção neste domínio:

1. ***"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.

2. "***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.

3. **"*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.

4. **"*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.

5. **"*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.

6. **"*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.

7. **"*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.

8. "***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.

9. **“*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.

10. **“*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.

11. **“*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.

3.3.2 DespesaException

O domínio “*DespesaException*” é responsável por gerenciar exceções relacionadas ao contexto de despesas no sistema. Cada subclasse específica aborda cenários distintos que podem levar a erros nesse contexto. Aqui estão as explicações para cada exceção dentro deste domínio:

1. “***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.

2. **"*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.

3. **"*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.

4. **"*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.

5. **"*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.

6. **"*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.

7. **"*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.

8. **"*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.

3.3.3 DocumentoException

O domínio "*DocumentoException*" é responsável por gerenciar exceções relacionadas ao contexto de documentos no sistema. Cada subclasse específica trata de situações

particulares que podem levar a erros dentro desse contexto. Aqui estão as explicações detalhadas para cada exceção neste domínio:

1. **"*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.

2. **"*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.

3. **"*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.

4. **"*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.

5. **“*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.

6. **“*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.

7. **“*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.

3.3.4 ImovelException

O domínio “*ImovelException*” abrange exceções relacionadas ao contexto de imóveis no sistema. Cada subclasse específica lida com situações distintas que podem resultar em erros no manejo de informações sobre imóveis. Abaixo estão explicações detalhadas para cada exceção neste domínio:

1. **“*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.

2. **"*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.

3. **"*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.

4. **"*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.

5. **"*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.

6. **"*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.

7. **"*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.

8. **"*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.

3.3.5 ProprietarioException

O domínio "*ProprietarioException*" trata de exceções relacionadas aos proprietários no sistema. Cada subclasse específica aborda cenários distintos que podem levar a erros no manuseio de informações sobre proprietários. Aqui estão as explicações detalhadas para cada exceção neste domínio:

1. **“*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.

2. **“*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.

3. **“*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.

4. **“*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.

5. **“*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.

6. **"*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.

7. **"*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.

8. **"*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.

3.3.6 ReuniaoException

O domínio "*ReuniaoException*" é responsável por exceções relacionadas às reuniões no sistema. Cada subclasse específica trata de situações particulares que podem levar a erros no gerenciamento de informações sobre reuniões. A seguir estão explicações detalhadas para cada exceção neste domínio:

1. **"*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.

2. **"*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.

3. **"*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.

4. **"*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.

5. **"*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.

6. **"*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.

7. **"*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.

8. **"*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.

3.4 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.

3.4.1 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:

1. **Nome do Condomínio:**

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.

2. **Endereço do Condomínio:**

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.

3. **Despesas do Condomínio:**

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.

4. **Imóveis do Condomínio:**

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.

5. **Proprietários do Condomínio:**

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.

6. **Reuniões do Condomínio:**

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.

7. **Documentos do Condomínio:**

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.

Essas regras visam garantir que instâncias da classe "*Condominio*" estejam em um estado consistente e atendam aos requisitos necessários para seu correto funcionamento no contexto do sistema. Se alguma condição não for satisfeita, a exceção correspondente será lançada, indicando qual regra de validação não foi cumprida.

3.4.2 DespesaRegras

A classe "*DespesaRegras*" contém um método estático chamado "*ValidarDespesa*", que recebe um objeto do tipo "*Despesa*" como parâmetro. O propósito deste método é validar se um objeto "*Despesa*" atende a determinadas regras de negócio. Abaixo estão as regras e suas respetivas condições:

8. **Tipo da Despesa:**

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.

9. **Imóvel Associado à Despesa:**

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.

10. **Valor da Despesa:**

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.

11. **Data de Vencimento da Despesa:**

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.

Estas regras ajudam a garantir que as instâncias da classe "*Despesa*" sejam consistentes e atendam aos requisitos necessários para o seu correto funcionamento no contexto do sistema em que são utilizadas. Similar ao caso anterior, se alguma condição não for satisfeita, a exceção correspondente é lançada indicando qual regra de validação não foi cumprida.

3.4.3 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:

12. **Tipo do Documento:**

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.

13. Nome do Documento:

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.

14. Conteúdo do Documento:

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.

Essas regras visam garantir que instâncias da classe "*Documento*" estejam em um estado consistente e atendam aos requisitos necessários para seu correto funcionamento no contexto do sistema. Se alguma condição não for satisfeita, a exceção correspondente será lançada, indicando qual regra de validação não foi cumprida.

3.4.4 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:

15. Endereço do Imóvel:

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.

16. Despesas Associadas ao Imóvel:

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.

17. <u>Proprietários Associados ao Imóvel:</u>

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.

Essas regras ajudam a garantir que as instâncias da classe "*Imovel*" estejam em um estado consistente e atendam aos requisitos necessários para o seu correto funcionamento no contexto do sistema em que são utilizadas. Da mesma forma que nas classes anteriores, se alguma condição não for satisfeita, a exceção correspondente é lançada indicando qual regra de validação não foi cumprida.

3.4.5 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:

18. <u>Nome do Proprietário:</u>

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.

19. <u>Contato do Proprietário:</u>

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.

20. Imóvel Associado ao Proprietário:

Regra: O proprietário deve estar associado a um imóvel.

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

21. NIF do Proprietário:

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.

Essas regras auxiliam na garantia de que instâncias da classe "*Proprietario*" estejam em um estado consistente e atendam aos requisitos necessários para seu correto funcionamento no contexto do sistema. Se alguma condição não for satisfeita, a exceção correspondente será lançada, indicando qual regra de validação não foi cumprida.

3.4.6 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:

22. Data da Reunião:

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.

23. Hora da Reunião:

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.

24. Local da Reunião:

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.

25. Intervenientes na Reunião:

Regra: Deve haver pelo menos um interveniente na reunião.

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

Essas regras asseguram que instâncias da classe "*Reuniao*" estejam em um estado consistente e atendam aos requisitos necessários para seu correto funcionamento no contexto do sistema. Se alguma condição não for satisfeita, a exceção correspondente será lançada, indicando qual regra de validação não foi cumprida.

3.5 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.

3.5.1 ICondominio

A interface "*ICondominio*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Ela estabelece um contrato para as classes que representam condomínios no sistema. Aqui está uma explicação detalhada de cada membro da interface:

26. <u>Propriedade "Nome":</u>

Tipo: "*string*".
Descrição: Obtém ou define o nome do condomínio.

27. <u>Propriedade "Endereco":</u>

Tipo: "*string*".
Descrição: Obtém ou define o endereço do condomínio.

28. <u>Propriedade "Despesas":</u>

Tipo: "*List<IDespesa>*".
Descrição: Obtém ou define a lista de despesas associadas ao condomínio.

29. <u>Propriedade "Imoveis":</u>

Tipo: "*List<IImovel>*".
Descrição: Obtém ou define a lista de imóveis associados ao condomínio.

30. <u>Propriedade "Proprietarios":</u>

Tipo: "*List<IProprietario>*".
Descrição: Obtém ou define a lista de proprietários associados ao condomínio.

31. <u>Propriedade "Reunioes":</u>

Tipo: "*List<IReuniao>*".

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

32. <u>Propriedade "Documentos":</u>

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.

Cada propriedade representa uma parte essencial da informação associada a um condomínio, como nome, endereço, despesas, imóveis, proprietários, reuniões e documentos.

O uso de interfaces facilita a criação de classes que podem ser tratadas de maneira uniforme no sistema, promovendo flexibilidade e consistência no código.

3.5.2 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:

33. <u>Propriedade "*Tipo*":</u>

Tipo: "*string*".

Descrição: Obtém ou define o tipo da despesa.

34. <u>Propriedade "*Valor*":</u>

Tipo: "*decimal*".

Descrição: Obtém ou define o valor da despesa.

35. Propriedade "*DataVencimento*":

Tipo: "*DateTime*".
Descrição: Obtém ou define a data de vencimento da despesa.

36. Propriedade "*EstadoPagamento*":

Tipo: "*bool*".
Descrição: Obtém ou define o estado de pagamento da despesa.

37. 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.

A implementação dessa interface em outras classes obriga essas classes a fornecerem implementações concretas para cada uma das propriedades listadas.

O uso de interfaces como essa ajuda na definição de contratos claros e na promoção da consistência e flexibilidade do código, permitindo que diferentes classes que representam despesas sejam tratadas de maneira uniforme no sistema.

3.5.3 IDocumento

A interface "*IDocumento*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Essa interface estabelece um contrato para as classes que representam documentos no sistema. Aqui está uma explicação detalhada de cada membro da interface:

38. Propriedade Nome:

Tipo: "*string*".
Descrição: Obtém ou define o nome do documento.

39. <u>Propriedade Tipo:</u>

Tipo: "*string*".
Descrição: Obtém ou define o tipo do documento.

40. <u>Propriedade Conteudo:</u>

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.

A implementação dessa interface em outras classes obriga essas classes a fornecerem implementações concretas para cada uma das propriedades listadas.

O uso de interfaces como essa ajuda na definição de contratos claros e na promoção da consistência e flexibilidade do código, permitindo que diferentes classes que representam documentos sejam tratadas de maneira uniforme no sistema.

3.5.4 IImovel

A interface "*IImovel*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Essa interface estabelece um contrato para as classes que representam imóveis no sistema. Aqui está uma explicação detalhada de cada membro da interface:

41. <u>Propriedade "*IdImovel*":</u>

Tipo: "*int*".
Descrição: Obtém ou define o identificador único do imóvel.

42. <u>Propriedade "*Proprietarios*":</u>

Tipo: "*List<IProprietario>*".

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

43. Propriedade "*Despesas*":

Tipo: "*List<IDespesa>*".

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

44. Propriedade "*Quotas*":

Tipo: "*List<decimal>*".

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

45. 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.

A implementação dessa interface em outras classes obriga essas classes a fornecerem implementações concretas para cada uma das propriedades listadas.

O uso de interfaces como essa ajuda na definição de contratos claros e na promoção da consistência e flexibilidade do código, permitindo que diferentes classes que representam imóveis sejam tratadas de maneira uniforme no sistema.

3.5.5 IProprietario

A interface "IProprietario" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Essa interface estabelece um contrato para as classes que representam proprietários no sistema. Aqui está uma explicação detalhada de cada membro da interface:

46. Propriedade "*Nome*":

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

47. Propriedade "*Contato*":

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

48. Propriedade "*Imovel*":

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

49. 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.

A implementação dessa interface em outras classes obriga essas classes a fornecerem implementações concretas para cada uma das propriedades listadas.

O uso de interfaces como essa ajuda na definição de contratos claros e na promoção da consistência e flexibilidade do código, permitindo que diferentes classes que representam proprietários sejam tratadas de maneira uniforme no sistema.

3.5.6 IReuniao

A interface "*IReuniao*" define um conjunto de propriedades que qualquer classe que a implemente deve fornecer. Essa interface estabelece um contrato para as classes que

representam reuniões no sistema. Aqui está uma explicação detalhada de cada membro da interface:

50. <u>Propriedade Data:</u>

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

51. <u>Propriedade Hora:</u>

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

52. <u>Propriedade Local:</u>

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

53. <u>Propriedade Intervenientes:</u>

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.

A implementação dessa interface em outras classes obriga essas classes a fornecerem implementações concretas para cada uma das propriedades listadas.

O uso de interfaces como essa ajuda na definição de contratos claros e na promoção da consistência e flexibilidade do código, permitindo que diferentes classes que representam reuniões sejam tratadas de maneira uniforme no sistema.

3.6 Gestor (Program)

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.

4. 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, buscamos 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, exploramos detalhadamente cada componente, examinando as decisões de design que fundamentam a arquitetura do sistema.

5. Referências