Parte Teórica do Projeto: Orçamento de Aluguel

Desafio R.M. Imobiliária - Estrutura Lógica, Código e Fluxograma

10 de novembro de 2025

1. Estrutura Lógica e Pensamento Algorítmico

O projeto foi desenvolvido em Python, utilizando o paradigma de **Programação Orientada a Objetos (POO)** para organizar as regras de cálculo e gerar relatórios automatizados. A aplicação segue um fluxo lógico bem definido: entrada de dados → processamento (cálculo das parcelas) → saída (valores e relatório).

1.1. Organização em Classes (POO)

As classes foram utilizadas para modelar o problema do aluguel de forma modular e reutilizável. Abaixo está um exemplo do uso de herança, encapsulamento e polimorfismo:

```
class Imovel:
    def __init__(self, tipo_imovel, valor_base, possui_garagem=False):
        self.tipo_imovel = tipo_imovel
        self.valor_base = valor_base
        self.possui_garagem = possui_garagem

    def calcular_adicionais(self):
        raise NotImplementedError("Subclasses devem implementar este método.")
```

➡ Nesta estrutura, a classe **Imovel** funciona como classe base, representando o conceito genérico de um imóvel. As subclasses especializadas (Apartamento, Casa e Estúdio) herdam dessa estrutura e reescrevem o método *calcular_adicionais()*, aplicando o conceito de **polimorfismo**.

2. Aplicação de Regras e Encapsulamento

As regras de cálculo são aplicadas dentro das subclasses. Cada tipo de imóvel possui suas próprias regras de preço, descontos e adicionais. Isso demonstra o uso de **encapsulamento** — cada classe é responsável apenas pelo que lhe compete.

```
class Apartamento(Imovel):
    def calcular_adicionais(self):
        adicional = 0.0
        if self.num_quartos == 2:
            adicional += 200.00  # Regra C
        if self.possui_garagem:
            adicional += 300.00  # Regra E
        self.mensalidade_adicionais = adicional
```

➡ Neste exemplo, a classe **Apartamento** aplica duas regras específicas: R$200 adicionais para 2 quartos e R$300 adicionais se o imóvel possuir garagem.

3. Geração de Orçamento e Pensamento Algorítmico

A lógica de geração do orçamento foi implementada na classe **OrcamentoGenerator**. Ela utiliza um algoritmo que percorre 12 meses e adiciona a parcela do contrato apenas nos 5 primeiros meses.

```
def _gerar_parcelas(self):
    parcelas = []
    for mes in range(1, 13):
        contrato = self.parcela_contrato if mes <= 5 else 0.0
        total = self.aluguel_mensal + contrato
        parcelas.append({'mes': mes, 'total': total})
    return parcelas
```

➡ Este algoritmo implementa o pensamento algorítmico, pois traduz regras de negócio (parcelamento em 5 meses) em uma estrutura lógica iterativa (loop de 1 a 12 meses).

4. Interface Gráfica (Front-End com Tkinter)

Para tornar a aplicação acessível, foi criada uma interface gráfica com o módulo **Tkinter**. O usuário pode selecionar o tipo de imóvel, informar adicionais e gerar o orçamento automaticamente.

```
class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Orçamento de Aluguel R.M.")
        self.geometry("750x450")
        self._setup_ui()
```

➡ A interface é construída com widgets como *LabelFrame*, *Button* e *Combobox*, permitindo que o usuário configure os parâmetros do cálculo sem precisar interagir com o terminal.

5. Conexão com os Requisitos da Entrega

O projeto atende todos os critérios definidos no edital do curso:

1. Vídeo Pitch (2,0 pts)	Demonstração em vídeo explicando o objetivo, código e in
2. Parte Teórica (2,0 pts)	Este documento apresenta o fluxograma, explicações e trechos de código coment
3. Parte Prática (4,0 pts)	Código-fonte funcional em Python com POO e Tkinter, pronto para publicação no