PUC Minas

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

GRUPO: Ana Julia da Silva, Guilherme Setragni Freitas, Milena Carlos da Silva, Natália Silva Oliveira, Pedro Cesar Ferreira Gomes.

Turma: Sistemas de Informação, 3º período
Professor(a): Charlene Cassia de Resende
Disciplina: Programação por Objetos

CLÍNICA MÉDICA: CLÍNICA VIDA

BETIM
2025

1. DESCRIÇÃO DETALHADA DO PROJETO

O sistema de Clínica Médica é uma aplicação de console desenvolvida em C# que simula as operações básicas de gerenciamento de pacientes, médicos e agendamento de consultas. Ele permite o cadastro, listagem, edição, exclusão e busca de informações sobre pacientes e médicos, além de funcionalidades para agendamento, listagem, edição, cancelamento e busca de consultas. O sistema também inclui relatórios básicos e filtragens. Os dados são persistidos em arquivos JSON.

1.1 PRINCIPAIS FUNCIONALIDADES

1.1.1 GESTÃO DE PACIENTES

- Cadastrar novos pacientes com nome, CPF, data de nascimento, telefone, email e histórico médico.
- Listar todos os pacientes cadastrados.
- Editar informações de pacientes existentes.
- Excluir pacientes do sistema.
- Buscar pacientes por nome ou CPF.
- Visualizar histórico de consultas de um paciente.

1.1.2 GESTÃO DE MÉDICOS

- Cadastrar novos médicos com nome, CPF, data de nascimento, telefone, email, especialidade e CRM.
- Listar todos os médicos cadastrados.
- Editar informações de médicos existentes.
- Excluir médicos do sistema.
- Buscar médicos por nome, CRM ou especialidade.
- Filtrar médicos por especialidade.

1.1.3 GESTÃO DE CONSULTAS

- Agendar novas consultas, associando um paciente, um médico e uma data/hora.
- Validar agendamentos (data no futuro, paciente não ter mais de uma consulta com o mesmo médico no dia, médico com no máximo 10 consultas no dia, médico sem consultas no mesmo horário).
- Listar todas as consultas agendadas.
- Editar informações de consultas (data/hora, observações, status).
- Cancelar consultas.
- Registrar atendimento de uma consulta, adicionando prontuário, diagnóstico e prescrições.
- Buscar consultas por paciente, médico ou data.

- Gerar relatório de total de consultas por médico.
- Listar consultas específicas de um médico.

2. ARQUITETURA DO SISTEMA

O sistema segue uma arquitetura simples, orientada a objetos, comum para aplicações de console de pequeno porte. Ele é estruturado em classes que representam as entidades do domínio (Pessoa, Paciente, Medico, Consulta) e uma classe principal (Program) que gerencia a lógica de negócio, a interação com o usuário (menu e entradas/saídas) e a persistência de dados.

2.1 CAMADAS LÓGICAS

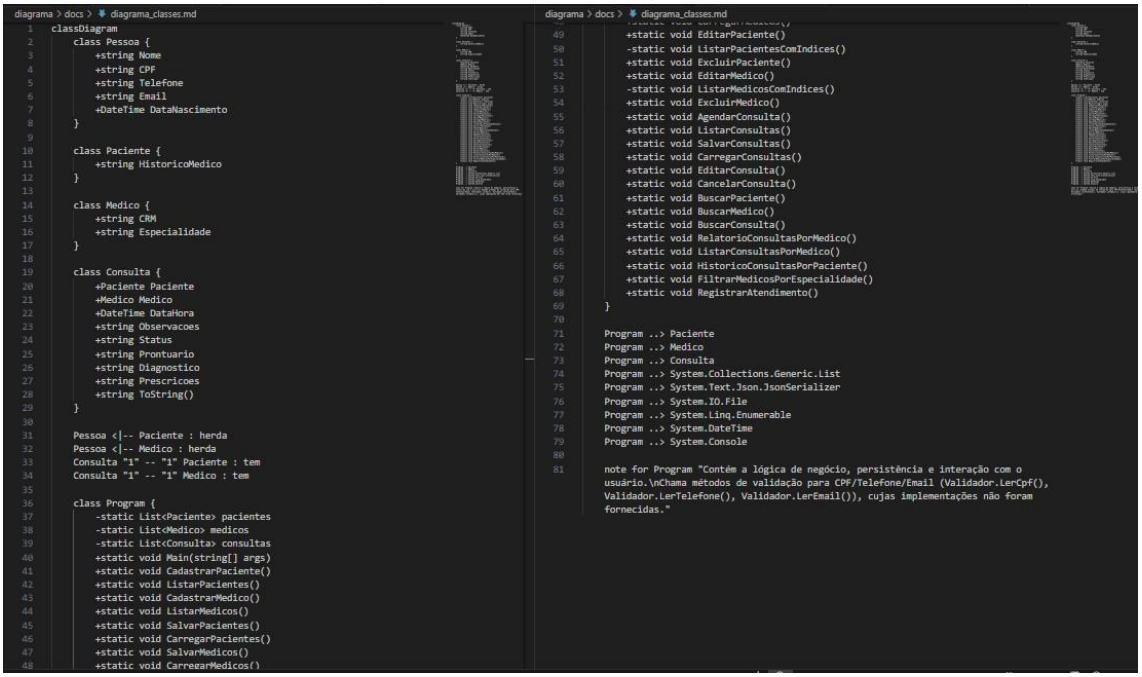
2.1.1 Camada de Apresentação/Interface do Usuário: Representada pelas interações do Console.WriteLine() e Console.ReadLine() dentro da classe Program, que lida com o menu e a entrada de dados do usuário.

2.1.2 Camada de Domínio/Modelo: Composta pelas classes Pessoa, Paciente, Medico e Consulta, que encapsulam os dados e comportamentos relacionados às entidades do negócio.

2.1.3 Camada de Persistência de Dados: Implementada diretamente na classe Program através das funções Salvar e Carregar que utilizam System.Text.Json para serializar e deserializar objetos para arquivos JSON (pacientes.json, medicos.json, consultas.json).

2.1.4 Camada de Lógica de Negócio/Serviço: As validações e regras de negócio (como validações de agendamento de consultas) estão embutidas nas funções da classe Program que manipulam as listas de objetos.

3. DIAGRAMA DE CLASSES



```
diagrama > docs > diagrama_classes.md
classDiagram
    class Pessoa {
        +string Nome
        +string CPF
        +string Telefone
        +string Email
        +DateTime DataNascimento
    }

    class Paciente {
        +string HistoricoMedico
    }

    class Medico {
        +string CRM
        +string Especialidade
    }

    class Consulta {
        +Paciente Paciente
        +Medico Medico
        +DateTime DataHora
        +string Observacoes
        +string Status
        +string Prontuario
        +string Diagnostico
        +string Prescricoes
        +string ToString()
    }

    Pessoa <|-- Paciente : herda
    Pessoa <|-- Medico : herda
    Consulta "1" -- "1" Paciente : tem
    Consulta "1" -- "1" Medico : tem

    class Program {
        -static List<Paciente> pacientes
        -static List<Medico> medicos
        -static List<Consulta> consultas
        +static void Main(string[] args)
        +static void CadastrarPaciente()
        +static void ListarPacientes()
        +static void CadastrarMedico()
        +static void ListarMedicos()
        +static void SalvarPacientes()
        +static void CarregarPacientes()
        +static void SalvarMedicos()
        +static void CarregarMedicos()
```

```
diagrama > docs > diagrama_classes.md
        +static void EditarPaciente()
        -static void ListarPacientesComIndices()
        +static void ExcluirPaciente()
        +static void EditarMedico()
        -static void ListarMedicosComIndices()
        +static void ExcluirMedico()
        +static void AgendarConsulta()
        +static void ListarConsultas()
        +static void SalvarConsultas()
        +static void CarregarConsultas()
        +static void EditarConsulta()
        +static void CancelarConsulta()
        +static void BuscarPaciente()
        +static void BuscarMedico()
        +static void BuscarConsulta()
        +static void RelatorioConsultasPorMedico()
        +static void ListarConsultasPorMedico()
        +static void HistoricoConsultasPorPaciente()
        +static void FiltrarMedicosPorEspecialidade()
        +static void RegistrarAtendimento()
    }

    Program ..> Paciente
    Program ..> Medico
    Program ..> Consulta
    Program ..> System.Collections.Generic.List
    Program ..> System.Text.Json.JsonSerializer
    Program ..> System.IO.File
    Program ..> System.Linq.Enumerable
    Program ..> System.DateTime
    Program ..> System.Console

    note for Program "Contém a lógica de negócio, persistência e interação com o
    usuário.\nChama métodos de validação para CPF/Telefone/Email (Validador.LerCpf(),
    Validador.LerTelefone(), Validador.LerEmail()), cujas implementações não foram
    fornecidas."
```

Explicação do Diagrama:

- **Pessoa (Classe Abstrata):** Contém as propriedades comuns a Pacientes e Médicos (Nome, CPF, Telefone, Email, DataNascimento).
- **Paciente:** Herda de Pessoa e adiciona a propriedade HistoricoMedico.
- **Medico:** Herda de Pessoa e adiciona as propriedades CRM e Especialidade.
- **Consulta:** Representa uma consulta médica, contendo referências a um Paciente e um
    - Medico, além de DataHora, Observacoes, Status, Prontuario, Diagnostico e Prescricoes. O método ToString() fornece uma representação textual da consulta.
- **Program:** A classe principal da aplicação. Ela gerencia as listas de pacientes, medicos e consultas, implementa o menu de interação e todas as funcionalidades do sistema, incluindo as operações CRUD e os relatórios. Também é responsável por salvar e carregar os dados em/de arquivos JSON.

4. PADRÕES DE PROJETO UTILIZADO

Devido à natureza da aplicação de console e à forma como a lógica está centralizada na classe Program, o uso de padrões de projeto formais é limitado. No entanto, podemos identificar alguns princípios e padrões implícitos:

4.1 Herança (Inheritance): Utilizado explicitamente com as classes Paciente e Medico herdando de Pessoa. Isso promove a reutilização de código e uma estrutura hierárquica clara para entidades com atributos comuns.

4.2 Singleton (Implícito/Global State): As listas pacientes, medicos e consultas são estáticas na classe Program, o que significa que há uma única instância dessas coleções acessível globalmente em toda a aplicação. Embora não seja um Singleton formalmente implementado com um construtor privado e método de instância única, o efeito prático é o de um estado global compartilhado. Para aplicações maiores, um padrão Singleton explícito ou injeção de dependência seria mais apropriado.

4.3 CRIU (Create, Read, Update, Delete): As operações básicas de manipulação de dados para Pacientes, Médicos e Consultas seguem o padrão CRUD.

4.4 Data Mapper (Implícito): A serialização e deserialização de objetos para JSON (usando System.Text.Json) e a leitura/escrita em arquivos (File.WriteAllText, File.ReadAllText) agem como uma forma simplificada de Data Mapper, convertendo objetos do domínio para uma representação persistente e vice-versa.