MANUAL E DESCRIÇÃO DO SOFTWARE

ELE078 Programação Orientada a Objetos, Turma TECAD

Trabalho Prático 2025/1

Matheus Marcondes de Oliveira (2022112517)

Sistema de Prontuário Eletrônico e Apoio à Decisão Clínica

Manual e Descrição do Software

Universidade Federal de Minas Gerais

Escola de Engenharia

Departamento de Engenharia Elétrica

ELE078 Programação Orientada a Objetos, Turma TECAD

Trabalho Prático 2025/1

Matheus Marcondes de Oliveira (2022112517)

Belo Horizonte, 22 de junho de 2025

Sumário

Nota do autor

Meu objetivo ao construir o MediClass não foi conceber uma solução revolucionária para uso em ambiente hospitalar real, tampouco elaborar um sistema superinteligente de complexidade elevada. Pelo contrário, proponho aqui uma idealização didática: aplicar de forma prática e contextualizada os princípios fundamentais da Programação Orientada a Objetos — herança, encapsulamento, polimorfismo e abstração — em um domínio que me desperta interesse, no caso, o fluxo de trabalho de uma instituição de saúde. O objetivo mor deste trabalho foi abstrair e implementar os conceitos aprendidos na sala de aula para uma situação completamente nova. Por isso também, busquei executar um projeto que, até onde tenho conhecimento, não foi proposto. Ao longo do desenvolvimento, repensei bastante sobre a complexidade do sistema que acabei propondo a mim mesmo, mas levo isso como um aprendizado. Muito ainda pode ser aperfeiçoado, mas notar as falhas nessa evolução foi um passo construtivo para mim.

Ao modelar as classes como Profissional, Paciente, Anamnese e Diagnóstico, busquei demonstrar como a POO facilita a organização e a estruturação de informações cotidianas, tornando o código mais legível, modular e pronto para evolução. Essa “simulação hospitalar” serve sobretudo como exercício acadêmico, reforçando o aprendizado dos conceitos vistos em sala e evidenciando seu valor na construção de sistemas coesos e de manutenção simplificada.

Introdução

Como forma de consolidar os conceitos aprendidos em Programação Orientada a Objetos ao longo deste semestre, desenvolveu-se um sistema de gestão de prontuários médicos capaz de registrar, organizar e preservar informações clínicas dos pacientes, atendendo a uma necessidade real em unidades de saúde.

O projeto explora os pilares da orientação a objetos: **encapsulamento, herança, abstração e polimorfismo** para modelar entidades como paciente, profissional de saúde e exame, garantindo código modular, extensível e de fácil manutenção.

Adicionalmente, com base na interpretação automatizada dos dados registrados (sinais vitais, histórico de sintomas, resultados de exames), implementou-se um módulo de sugestão de diagnóstico que, a partir de regras de decisão clínica, auxilia o profissional de saúde na identificação de possíveis patologias.

Essa funcionalidade demonstra como os padrões de projeto e a reutilização de código podem habilitar a criação de ferramentas de apoio à decisão, alinhando o aprendizado acadêmico com aplicações práticas na área da saúde.

O sistema

O MediClass está estruturado em seis módulos principais que, em conjunto, formam um sistema de prontuário eletrônico e apoio à decisão clínica. O **main.py** atua como ponto de entrada, **sistema.py** orquestra o fluxo de execução via linha de comando, **profissionais.py** define os perfis de usuários (Profissional, Médico, Enfermeiro e Técnico), **paciente.py** modela o paciente e gerencia seu histórico, **anamnese.py** cuida da coleta de sinais vitais e categorização de sintomas, e **diagnostico.py** encapsula as sugestões clínicas geradas pelo médico.

No **main.py**, todo o ciclo de vida da aplicação é iniciado e encerrado. As funções load_data() e save_data() garantem a persistência de usuários e pacientes em JSON, permitindo que dados sejam recarregados em execuções subsequentes. Em seguida, instancia-se o objeto SistemaMediclass, registram-se usuários-padrão para testes e, finalmente, chama-se sistema.executar(), que dá início ao loop interativo main.

Já o **sistema.py** concentra a lógica de interface: ao instanciar SistemaMediclass, criam-se dois dicionários em memória (usuarios e pacientes). O método login() solicita credenciais, autentica chamando Profissional.autenticar() e retorna o objeto correspondente ou None. Uma vez logado, menu_principal() exibe opções (registro de pacientes, triagem, consulta, visualização, exportação, adição de exames) e despacha cada escolha para métodos, que validam permissões via isinstance antes de delegar a ação ao profissional apropriado.

O módulo **profissionais.py** centraliza os perfis de usuário. A superclasse Profissional encapsula dados comuns (nome, registro, login e hash de senha) e fornece o método _hash_senha() e autenticar() para controle de acesso. As subclasses especializam esse comportamento:

- **Enfermeiro** implementa triagem(paciente: Paciente), que guia a coleta de frequência cardíaca, pressão arterial, oximetria e respostas específicas por tipo de sintoma;
- **Medico** adiciona sugerir_diagnosticos(), gerar_receituario() e gerar_declaracao_comparecimento(), produzindo relatórios em TXT;

- **Tecnico** oferece adicionar_exame_sistema(), permitindo selecionar tipos de exame predefinidos e registrar seus resultados no paciente .

Em **paciente.py**, a classe Paciente modela todas as informações de um paciente: dados cadastrais (nome, CPF, convênio, data de nascimento, leito), data de entrada, nome do enfermeiro que realizou a triagem, lista de resultados de exames e flag de prioridade. O histórico é persistido em arquivo historicos/{cpf}.txt, e métodos como registrar_entrada(), atualizar_historico() e consultar_historico() garantem a consistência e rastreabilidade de cada evento clínico paciente.

O **anamnese.py** define o enum TipoSintoma (gastrointestinal, respiratório, cardiovascular, trauma, dermatológico e outros) e a classe Anamnese, que armazena frequência cardíaca, pressão arterial, saturação de O_2, respostas sim/não, tipo e detalhes de sintoma, além de timestamp. A interface pública é o método to_dict(), que serializa todos esses dados para fácil impressão ou exportação anamnese.

Por fim, o **diagnostico.py** encapsula cada sugestão clínica no objeto Diagnostico, que guarda categoria, descricao e uma lista de exames_sugeridos. O método __str__() formata a recomendação para exibição no terminal, fechando o ciclo de decisão clínica iniciado na anamnese diagnostico.

A interface

A interface do **SistemaMediclass** é totalmente baseada em linha de comando: todas as interações ocorrem por meio de prompts exibidos com print() e capturados via input(), desde a autenticação de login e senha até a seleção de opções no menu principal e o fornecimento de dados de pacientes (CPF, nome, convênio, respostas S/N etc.). O loop do menu principal é reexibido continuamente até o usuário optar por “0. Logout”, e cada escolha é despachada para métodos que verificam o tipo de profissional (Enfermeiro, Médico ou Técnico) antes de executar ações como triagem, diagnóstico, exportação de prontuário ou registro de exames.

Repositório

Os arquivos do sistema, readme.md e documentação estão disponíveis no GitHub:

https://github.com/matheusmarcondes1/mediclass

Fluxograma do sistema

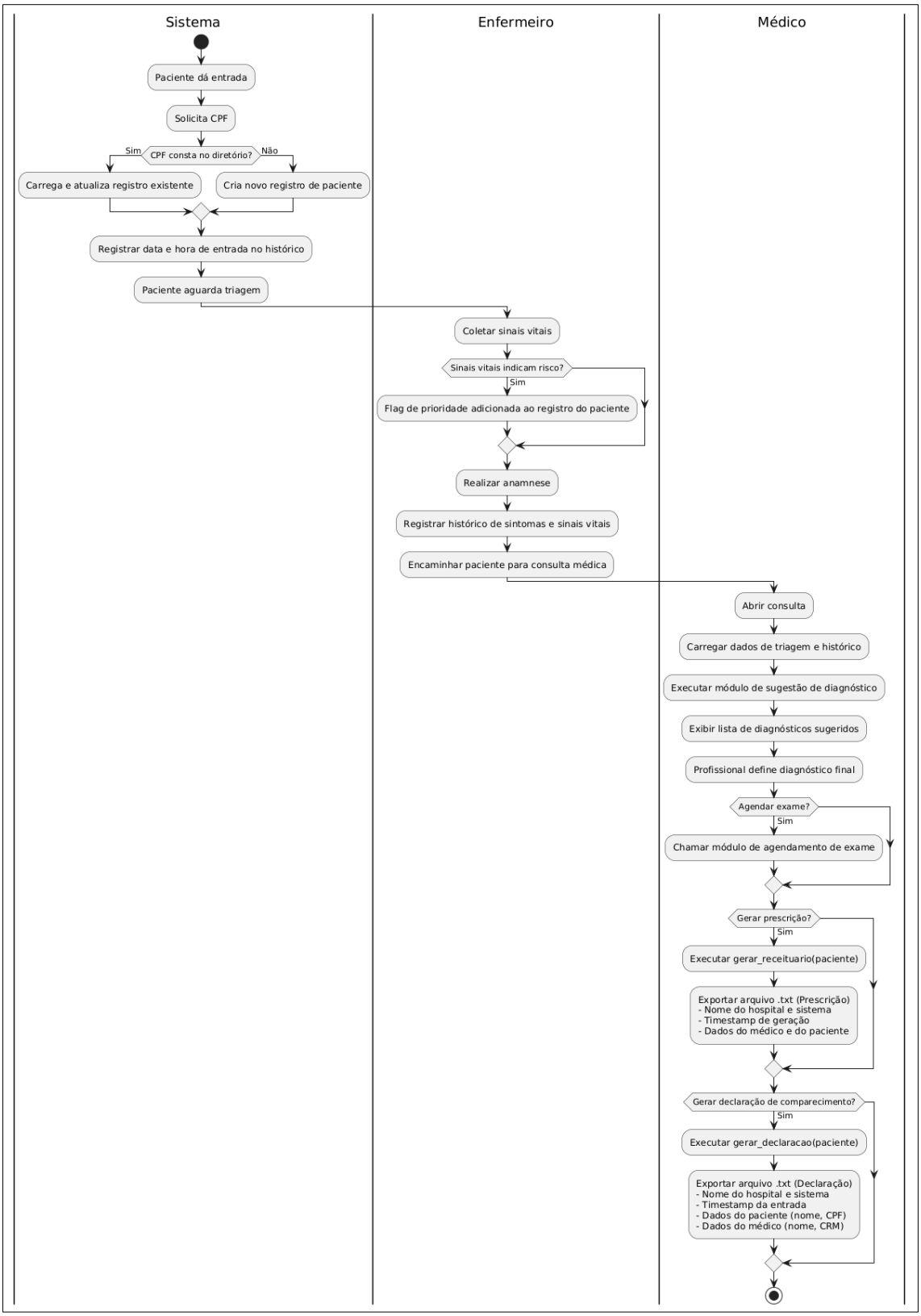


Fig. 1 – Fluxograma de um paciente no sistema

Demonstração

Entrada de paciente

Mensagem do sistema no prompt de comando	Resposta do usuário + Enter
Login: Senha	enf senha
Bem-vindo(a), Enf. Teste! --- Menu Principal --- 1. Registrar entrada de paciente 2. Realizar triagem (enfermeiro) 3. Realizar consulta (médico) 4. Visualizar prontuário 5. Exportar prontuário (.txt) 6. Adicionar exame (técnico) 0. Logout Escolha uma opção:	1
CPF do paciente:	15227583641
Paciente não encontrado. Cadastrando novo. Nome completo:	Matheus Marcondes de Oliveira
Contato:	31984950000
Convênio:	Unimed
Data de nascimento (YYYY-MM-DD):	2003-06-01
Leito	1A
Entrada registrada.	
--- Menu Principal --- 1. Registrar entrada de paciente 2. Realizar triagem (enfermeiro) 3. Realizar consulta (médico) 4. Visualizar prontuário 5. Exportar prontuário (.txt) 6. Adicionar exame (técnico) 0. Logout Escolha uma opção:	

Triagem

Login: Senha	enf senha
Bem-vindo(a), Enf. Teste! --- Menu Principal --- 1. Registrar entrada de paciente 2. Realizar triagem (enfermeiro) 3. Realizar consulta (médico) 4. Visualizar prontuário 5. Exportar prontuário (.txt) 6. Adicionar exame (técnico) 0. Logout Escolha uma opção:	2
CPF do paciente:	15227583641
Frequência cardíaca (40-200 bpm):	70
Pressão arterial sistólica (70-250 mmHg):	250
Pressão sistólica incomum! Ativando prioridade.	
Pressão arterial diastólica (40-150 mmHg):	90
Oximetria de pulso (85-100%):	99
Selecione o tipo de sintoma: 1. gastrointestinal 2. respiratório 3. cardiovascular 4. trauma 5. dermatológico 6. outros	1
Náuseas ou vômitos (S/N): Diarreia (S/N): Constipação (S/N): Dor abdominal (S/N): Perda de peso inexplicada (S/N): Febre (S/N):	S S S S S S
Triagem concluída.	
--- Menu Principal --- 1. Registrar entrada de paciente 2. Realizar triagem (enfermeiro) 3. Realizar consulta (médico) 4. Visualizar prontuário 5. Exportar prontuário (.txt) 6. Adicionar exame (técnico) 0. Logout Escolha uma opção:	

Consulta

Login: Senha	med senha
Bem-vindo(a), Dr. Teste! --- Menu Principal --- 1. Registrar entrada de paciente 2. Realizar triagem (enfermeiro) 3. Realizar consulta (médico) 4. Visualizar prontuário 5. Exportar prontuário (.txt) 6. Adicionar exame (técnico) 0. Logout Escolha uma opção:	2
CPF do paciente:	15227583641
Categoria: Gastrointestinal 1. Desconforto abdominal 2. Disfunção intestinal 3. Dor aguda Selecione o subgrupo (1-3): 2	2
Febre + náuseas + diarreia? (S/N):	S
--- Diagnósticos sugeridos --- Gastrointestinal: Possível gastroenterite infecciosa (Exames sugeridos: Hemograma, Coprocultura)	
Deseja solicitar exame a um técnico? (S/N):	S
Solicitação enviada.	
Deseja gerar receituário? (S/N):	S
Iniciando prescrição. Digite 0 para finalizar. Nome da medicação (ou 0 para terminar):	Ciprofloxacino
Posologia [miligramas]:	500
Intervalo das doses [horas]:	12
Período de tratamento [dias]:	5
Nome da medicação (ou 0 para terminar):	Lactobacillus acidophilus
Posologia [miligramas]:	2
Intervalo das doses [horas]:	24
Período de tratamento [dias]:	14
Nome da medicação (ou 0 para terminar):	0
Receituário exportado para receituario_15227583641.txt	
Deseja gerar declaração de comparecimento? (S/N):	S
Declaração exportada para declaracao_15227583641.txt	
Consulta encerrada. Retornando ao menu.	

Na consulta foram gerados os seguintes documentos:

```
receituario_15227583641.txt

Hospital da Escola de Engenharia da UFMG
Sistema MediClass
Prescrição gerada em 2025-06-23 12:26

Paciente: Matheus Marcondes de Oliveira (CPF: 15227583641)
Médico: Dr. Teste (CRM: CRM123)

- Ciprofloxacino: 500mg a cada 12 horas, durante 5 dias;
- Lactobacillus acidophilus: 2mg a cada 24 horas, durante 14 dias;


Dr. Teste - CRM CRM123
```

```
declaracao_15227583641.txt

Hospital da Escola de Engenharia da UFMG
Sistema MediClass
Prescrição gerada em 2025-06-23 12:26

Eu, Dr. Teste, CRM CRM123, atesto para os devidos fins que o paciente Matheus Marcondes
de Oliveira, CPF 15227583641, compareceu a consulta clínica no dia 2025-06-23, dando
entrada no hospital às 12:27, apresentando quadro sintomático gastrointestinal.

Cordialmente,
Dr. Teste - CRM CRM123
```

Exportação de prontuário

Login: Senha	med senha
Bem-vindo(a), Dr. Teste! --- Menu Principal --- 1. Registrar entrada de paciente 2. Realizar triagem (enfermeiro) 3. Realizar consulta (médico) 4. Visualizar prontuário 5. Exportar prontuário (.txt) 6. Adicionar exame (técnico) 0. Logout Escolha uma opção:	2
CPF do paciente:	15227583641
Prontuário exportado para prontuario_15227583641.txt	
--- Menu Principal --- 1. Registrar entrada de paciente 2. Realizar triagem (enfermeiro) 3. Realizar consulta (médico) 4. Visualizar prontuário 5. Exportar prontuário (.txt) 6. Adicionar exame (técnico) 0. Logout Escolha uma opção:	

O seguinte documento foi gerado a partir da exportação:

```
prontuario_15227583641.txt

Prontuário de Matheus Marcondes de Oliveira
CPF: 15227583641
Contato: 31984950000
Convênio: Unimed
Data de nascimento: 2003-06-01
Leito: 1A
Enfermeiro: Enf. Teste
Prioritário: Sim # Flag adicionada pelo valor de pressão sistólica incomum (250 mmHg)

Histórico Médico:
Histórico de Matheus Marcondes de Oliveira (CPF: 15227583641)
Criado em: 2025-06-23 12:08:50
--------------------------------------------------
[2025-06-23 12:08:50] Entrada no leito 1A em 2025-06-23
[2025-06-23 12:09:58] Triagem: {'frequencia_cardiaca': 70, 'pressao_arterial':
'250/90', 'saturacao_o2': 99, 'respostas_sim_nao': {'dor torácica opressiva': True,
'palpitações': True, 'tontura ou desmaio': True, 'edema de membros inferiores': True},
'tipo_sintoma': 'cardiovascular', 'detalhes_sintoma': None, 'timestamp': '2025-06-23
12:09:58'}
[2025-06-23 12:09:58] FLAG: Prioridade ativada devido a valores incomuns.
[2025-06-23 12:12:54] Entrada no leito 1A em 2025-06-23
[2025-06-23 12:13:39] Triagem: {'frequencia_cardiaca': 70, 'pressao_arterial':
'250/90', 'saturacao_o2': 99, 'respostas_sim_nao': {'náuseas ou vômitos': True,
'diarreia': True, 'constipação': True, 'dor abdominal': True, 'perda de peso
inexplicada': True, 'febre': True}, 'tipo_sintoma': 'gastrointestinal',
'detalhes_sintoma': None, 'timestamp': '2025-06-23 12:13:39'}
```

```
[2025-06-23 12:13:39] FLAG: Prioridade ativada devido a valores incomuns.
[2025-06-23 12:19:36] Solicitação de exame enviada ao técnico.
[2025-06-23 12:24:31] Prescrição adicionada em 2025-06-23 12:24: Ciprofloxacino, 500,
12, 5
[2025-06-23 12:26:21] Prescrição adicionada em 2025-06-23 12:26: Lactobacillus
acidophilus, 2, 24, 14
[2025-06-23 12:27:24] Declaração de comparecimento gerada em 2025-06-23 às 12:27.


Última Anamnese:
frequencia_cardiaca: 70
pressao_arterial: 250/90
saturacao_o2: 99
respostas_sim_nao: {'náuseas ou vômitos': True, 'diarreia': True, 'constipação': True,
'dor abdominal': True, 'perda de peso inexplicada': True, 'febre': True}
tipo_sintoma: gastrointestinal
detalhes_sintoma: None
timestamp: 2025-06-23 12:13:39
```

Falha: autenticação mal-sucedida

Ao entrar com um login não registrado, o sistema deve rejeitar o acesso e ser encerrado.

Login:	med
Senha:	123
Credenciais inválidas. Encerrando sistema.	

Falha: tentativa de acesso a método protegido por usuário não qualificado

Quando um usuário tenta acessar um método restrito à sua classe, o sistema deverá rejeitar o acesso e retornar ao menu.

No nosso exemplo, um técnico tentará iniciar uma triagem, que apenas pode ser realizada por objeto da classe Enfermeiro.

Login:	tec
Senha:	senha
Bem-vindo(a), Tec. Teste! --- Menu Principal --- 1. Registrar entrada de paciente 2. Realizar triagem (enfermeiro) 3. Realizar consulta (médico) 4. Visualizar prontuário 5. Exportar prontuário (.txt) 6. Adicionar exame (técnico) 0. Logout Escolha uma opção:	2
Acesso negado. Apenas enfermeiros podem realizar triagem.	
--- Menu Principal --- 1. Registrar entrada de paciente 2. Realizar triagem (enfermeiro) 3. Realizar consulta (médico) 4. Visualizar prontuário 5. Exportar prontuário (.txt) 6. Adicionar exame (técnico) 0. Logout Escolha uma opção:	

Falha: data de nascimento inválida

...	
Data de nascimento (YYYY-MM-DD):	1
Formato inválido. Use YYYY-MM-DD.	

Classes

Classe Paciente

A classe **Paciente** é a representação, em código, de cada paciente dentro do sistema Mediclass. Ela encapsula tanto os dados cadastrais (nome, CPF, contato, convênio, data de nascimento, leito e enfermeiro responsável na triagem) quanto o estado clínico e administrativo (data de entrada, histórico de exames, flag de prioridade).

Para dar suporte à persistência do histórico médico, toda vez que é chamado um método de registro ou atualização, essas informações são serializadas em um arquivo texto dentro da pasta historicos/, cujo nome é o próprio CPF do paciente.

Atributos

- Dados pessoais e de contato (nome, cpf, contato, convenio, data_nascimento, leito, enfermeiro_triagem)
- Estado dinâmico (data_entrada, resultados_exames, prioritario)
- Caminho do arquivo de histórico (_historico_file), gerado automaticamente em historicos/{cpf}.txt

Métodos

- registrar_entrada(): define a data de entrada como a data atual e grava um registro no histórico;
- atualizar_historico(registro: str): recebe uma string de evento (entrada, exame, prescrição etc.) e anexa ao arquivo de histórico com timestamp;
- consultar_historico() -> str: lê e retorna todo o conteúdo do arquivo de histórico;
- adicionar_exame(exame: str, resultado: str): adiciona o exame à lista resultados_exames e registra no histórico.

Diagrama UML

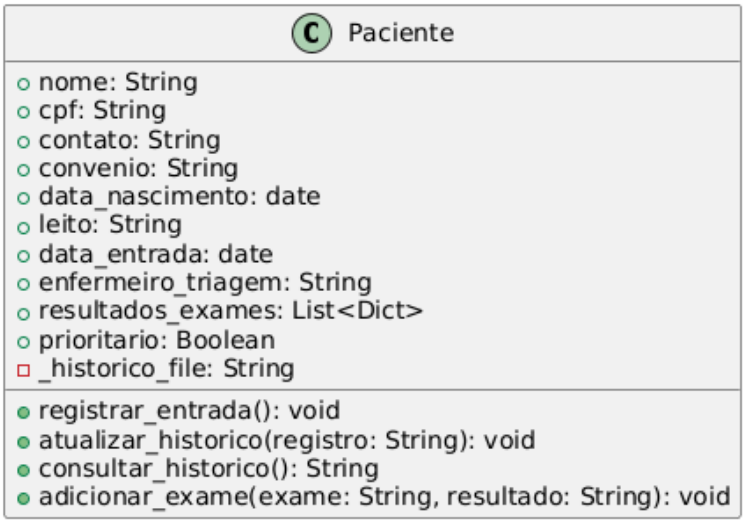


Fig. 2 – Diagrama UML da classe Paciente

Dependências

import os: módulo da biblioteca padrão que fornece funções para interagir com o sistema de arquivos. Em paciente.py ele é utilizado para garantir a criação e o acesso ao arquivo de histórico médico de cada paciente, em três situações:

1. os.makedirs('historicos', exist_ok=True) para garantir que a pasta "historicos" existe antes de gravar qualquer arquivo;
2. self._historico_file = os.path.join('historicos', f"{self.cpf}.txt") para criar o caminho do arquivo a ser gravado;
3. if not os.path.exists(self._historico_file):
   with open(self._historico_file, 'w', encoding='utf-8') as f: [...] para checar a pré-existência do arquivo a ser gravado, a fim de evitar sobrescrição.

from datetime import date, datetime: módulo que formata data e horário para timestamps no histórico.

Classe abstrata Profissional

A classe **Profissional** atua como superclasse para todos os usuários "profissionais" do sistema (médicos, enfermeiros e técnicos), encapsulando as informações e comportamentos comuns a eles.

Atributos

- nome: str # nome completo do profissional
- registro_profissional: str # número do CRM/COREN/CRTR, etc.
- login: str # identificador para acesso ao sistema
- _senha_hash: str # hash SHA-256 da senha, protegido

Métodos

_hash_senha(senha: str) -> str: função protegida para garantir que a senha nunca seja armazenada em texto simples.

autenticar(senha: str) -> bool: compara o hash da senha fornecida com _senha_hash, retornando True em caso de coincidência (login) e False caso contrário (encerra o sistema).

adicionar_paciente(paciente: Any) -> None: método abstrato (NotImplementedError), que força subclasses (e.g. Enfermeiro, Medico, Tecnico) a implementarem a lógica de cadastro ou associação de pacientes.

editar_paciente(paciente: Any) -> None: outro método abstrato para edição de dados do paciente, também concretizado nas subclasses.

Diagrama UML

Profissional
+ nome : str + registro_profissional : str + login : str # _senha_hash : str

_hash_senha(senha: str): str + autenticar(senha: str): bool + adicionar_paciente(paciente: Any) {abstract} + editar_paciente(paciente: Any) {abstract}

Dependências

import hashlib: módulo da biblioteca padrão para criptografia de hashes. Em profissionais.py, é usado em self._senha_hash = hashlib.sha256(senha.encode('utf-8')).hexdigest() no método _hash_senha, e em autenticar() para verificar senhas usando SHA-256[1].

from typing import Any: fornece o tipo genérico Any para anotações. Em Profissional, é usado nas assinaturas indicando que esses métodos podem receber qualquer objeto, para receber instâncias de Paciente.

from enum import Enum: classe base para criar enumerações. Em profissionais.py, serve para definir a classe ExamType (dentro da subclasse Tecnico) que lista todos os tipos de exame que o técnico pode registrar.

from datetime import date, datetime: módulo que formata data e horário para timestamps no histórico.

from datetime import date, datetime: formatação de data e hora, utilizada em timestamps nos registros gerados dentro dos métodos gerar_receituario() e gerar_declaracao_comparecimento() da subclasse Medico.

from paciente import Paciente: interação com o histórico de um paciente

from anamnese import Anamnese: respostas da anamnese do paciente

from anamnese import TipoSintoma: enumera classificadores sintomáticos. É utilizado pelos métodos Medico.sugerir_diagnosticos(), Medico.gerar_receituario() e Enfermeiro.triagem().

from diagnostico import Diagnostico: encapsula um diagnóstico e recomendações de exames.

Subclasse Medico

A classe **Medico** estende a classe abstrata **Profissional**, especializando-a para o contexto de um médico no sistema. Ela é responsável por analisar os dados clínicos do paciente, sugerir diagnósticos e gerar documentos oficiais (prescrição e declaração).

Herança e atributos

Herda de **Profissional**: nome, registro_profissional, login e _senha_hash (hash da senha) para autenticação e identificação no sistema.

[1] Um algoritmo criptográfico de hash que gera um resumo (hash) de tamanho fixo de 256 bits (32 bytes) de qualquer entrada de dados. Não foi abordado com detalhes pois não é o foco do trabalho.

Métodos

sugerir_diagnosticos(paciente: Paciente) → List[Diagnostico]: Lê a última anamnese registrada no paciente (paciente.ultima_anamnese), aplicando as regras da classe Diagnostico (definidas em diagnostico.py) para instanciar objetos Diagnostico(categoria, descrição, exames). Faz uma série de perguntas para uma consulta guiada com base no tipo de sintoma informado na anamnese, processa as respostas (booleanas) à essas perguntas em um sistema de decisão baseado em condições[2], e então retorna uma lista de sugestões que o médico pode revisar.

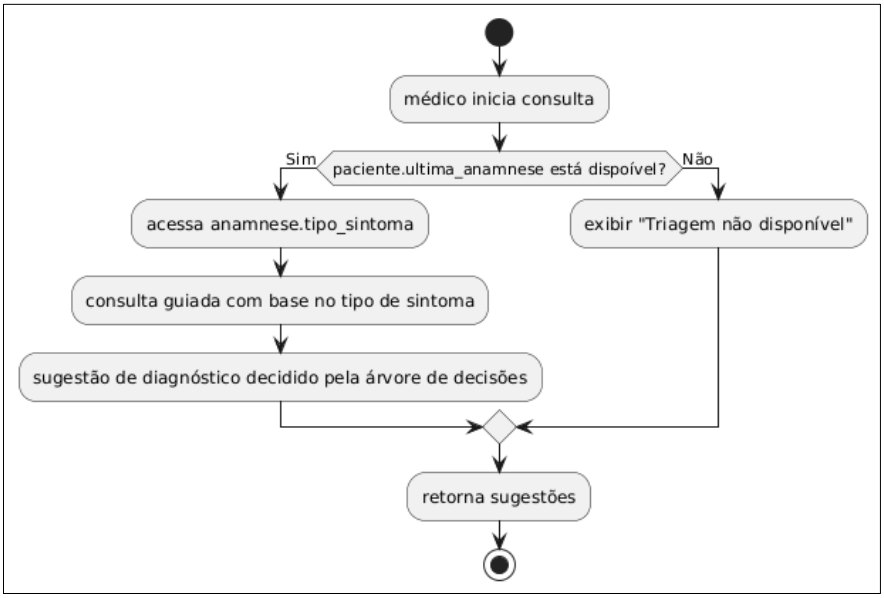


Fig. 3 – Fluxograma do método Medico.sugerir_diagnostico()

gerar_receituario(paciente: Paciente) → None: Gera um arquivo .txt ao final da consulta contendo: cabeçalho padronizado (nome do hospital, marca do sistema e timestamp, com fácil personalização) e dados do paciente e do médico. Além disso, entra em um loop para que o médico possa inserir medicações, dosagens, intervalo, período de tratamento. O loop é encerrado quando o médico insere “0” no campo do nome do medicamento.

gerar_declaracao_comparecimento(paciente: Paciente) → None: Gera uma declaração de comparecimento (atestado), a ser exportado em .txt, contendo:

- Nome completo do médico e CRM.
- Dados do paciente, com data/hora de registro e tipo de sintoma.
- Texto formal de atestado para os devidos fins.

[2] Este é o sistema de decisão clínica, responsável por retornar o diagnóstico, que funciona através de uma série de condições, denominada “árvore de decisões”, que está disponível no Apêndice A deste documento.

Diagrama UML

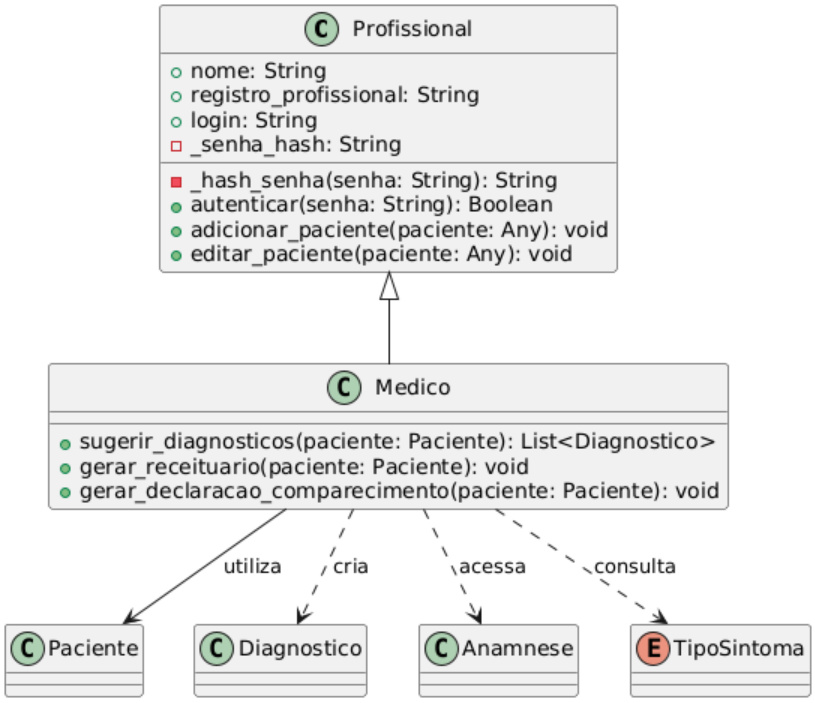


Fig. 4 – Diagrama UML da subclasse Medico(Profissional)

Subclasse Enfermeiro

A classe **Enfermeiro** especializa a superclasse **Profissional** para representar o profissional de enfermagem responsável pela triagem inicial dos pacientes.

Herança e atributos

Herda de **Profissional**: nome, registro_profissional, login e _senha_hash para autenticação.

Métodos

triagem(paciente: Paciente) → **None**: Coleta sinais vitais de forma guiada e realiza anamnese guiada por perguntas de resposta booleana.

O método instancia um objeto Anamnese(frequencia, pressao, oxigenacao, tipo_sintoma, detalhes, timestamp) usando o enum TipoSintoma para categorizar o sintoma. O objeto Anamnese é descrito pela classe Anamnese, implementada no módulo anamnese.py.

O procedimento de anamnese foi baseado em informações que constam na literatura, e contém perguntas que são classificadas em três tipos:

- **Sinais vitais**
  - Frequência cardíaca;
  - Pressão arterial (sistólica e diastólica);
  - Oximetria de pulso.
- **Tipo de sintoma**
  - Gastrointestinal;
  - Respiratório;
  - Cardiovascular;

    - Trauma;
    - Dermatológico;
    - Outros.
- **Específicas para o tipo de sintoma**
    - Duas perguntas para cada tipo de sintoma informado no início.

Para os sinais vitais, as respostas válidas são apenas valores int, dentro de uma janela ampla. Dentro dessa janela de valores válidos, existem ainda os valores comuns. Caso o paciente apresente um valor inválido, o sistema requererá que entre com um valor válido. Caso entre com um valor válido, mas incomum (ou seja, que apresente risco) será ativada uma flag de prioridade para aquele paciente, alterando o valor da variável booleana Paciente.prioritario (inicialmente declarada como False) para True, sendo registrado no histórico.

Sinal vital	Valores válidos		Valores comuns[3]	
F.C. [bpm]	40	200	70	120
Pressão S. [mmHg]	70	250	90	140
Pressão D. [mmHg]	40	150	60	90
O_2 [%]	85	100	95	100

O tipo de sintoma é enumerado na classe TipoSintoma em anamnese.py. Após as aferições de sinais vitais, um loop enumera e apresenta os tipos de sintomas válidos em um menu, e requer uma entrada do tipo int para selecioná-la. Apenas será aceita uma resposta do tipo int dentre 1 e o a maior opção (no caso, 8), inclusos, e continuará pedindo até que um valor válido seja informado, informando quando o requisito não é cumprido.

Após informar o tipo do sintoma, o método determina um conjunto de 2 perguntas apropriadas para cada situação. Estas são de afirmativa/negativa, onde o usuário deverá responder com um S para sim, ou N para não. A string é convertida automaticamente para maiúscula, para que tanto minúscula quanto maiúscula sejam opções interpretáveis.

Por fim, sobrescreve paciente.ultima_anamnese e chama paciente.atualizar_historico(anamnese.to_dict()) para registrar no histórico.

O método pode ser interpretado pelo seguinte fluxograma:

[3] Os valores comuns foram determinados com base em literatura.

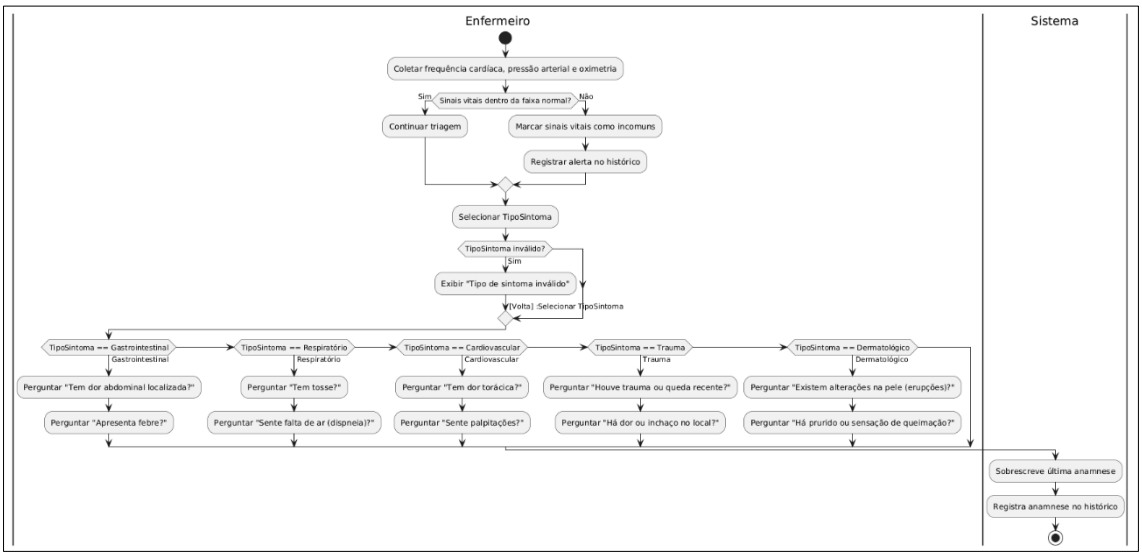


Fig. 5 – Fluxograma do método Enfermeiro.anamnese()

Diagrama UML

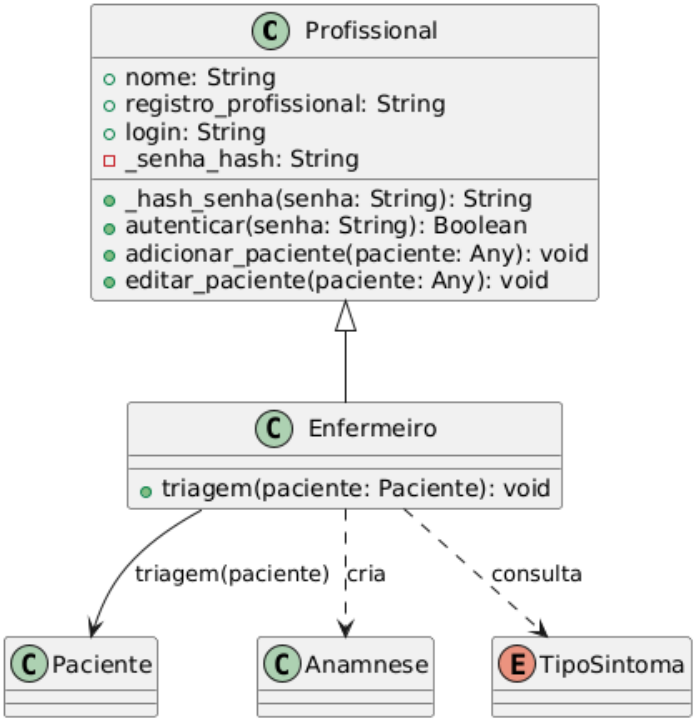


Fig. 6 – Diagrama UML da subclasse Enfermeiro(Profissional)

Subclasse Tecnico

A classe Tecnico especializa a superclasse Profissional para representar o técnico responsável pelo registro de exames no sistema.

Herança e atributos

Herda de **Profissional**: nome (str), registro_profissional (str), login (str) e _senha_hash (str) para autenticação.

Métodos

adicionar_exame_sistema(paciente: Paciente) → None:

Exibe ao usuário uma lista de tipos de exame enumerados no próprio módulo (por exemplo, hemograma, radiografia, ultrassonografia). Então, recebe a escolha do exame, por um menu enumerado e solicita o resultado correspondente (texto ou numérico).

Depois, chama paciente.adicionar_exame(exame, resultado) para atualizar o objeto e atualizar o histórico, adicionando o evento.

Diagrama UML

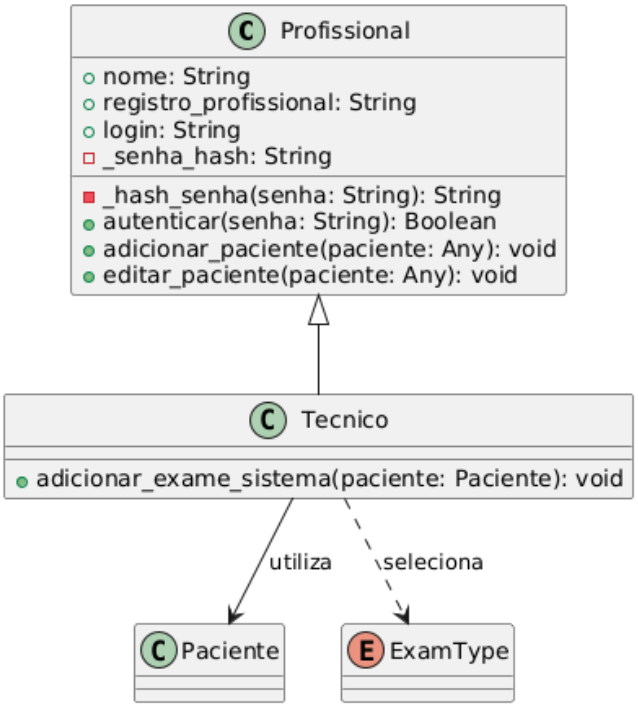


Fig. 7 – Diagrama UML da subclasse Tecnico(Profissional)

Classe TipoSintoma

TipoSintoma é uma enumeração que define as categorias clínicas usadas no processo de anamnese e da consulta clínica para padronizar a coleta de informações.

1. GASTROINTESTINAL: sintomas relacionados ao trato digestivo;
2. RESPIRATÓRIO: inclui manifestações como tosse, dispneia e dor torácica;
3. CARDIOVASCULAR: engloba angina, palpitações e dispneia de esforço;
4. TRAUMA: abrange queixas pós-trauma ou quedas, fraturas e hemorragia;
5. DERMATOLÓGICO: cobre alterações cutâneas;
6. OUTROS: categoria genérica para sinais e sintomas que não se enquadram nos grupos acima.

Integrações

Triagem: Em Enfermeiro.triagem(): o profissional escolhe um valor de TipoSintoma para determinar quais perguntas específicas serão feitas ao paciente.

Consulta: Na geração de diagnósticos via Medico.sugerir_diagnosticos(), o valor orienta a aplicação das regras clínicas em diagnostico.py. Além disso, ele constará no arquivo gerado de declaração de comparecimento.

Classe Anamnese

A classe **Anamnese** encapsula todos os dados coletados na triagem de enfermagem, estruturando sinais vitais, categoria de sintoma e contexto clínico em um único objeto, além de oferecer serialização para persistência.

Atributos

- frequencia_cardiaca: int
- pressao_arterial: str # ex.: "120/80" sistólica / diastólica
- oximetria: float # saturação de O_2
- tipo_sintoma: TipoSintoma # enumeração das categorias clínicas
- detalhes: str # respostas abertas sobre o sintoma
- timestamp: datetime # momento da coleta

Métodos

to_dict() -> Dict: Converte o objeto em dicionário, pronto para serialização em JSON ou gravação em arquivo.

Integrações

Triagem: Enfermeiro.triagem() instancia Anamnese com os valores coletados e atribui a paciente.ultima_anamnese, em seguida persiste no histórico via paciente.atualizar_historico(anamnese.to_dict()).

Consulta: Medico.sugerir_diagnosticos() lê paciente.ultima_anamnese para aplicar regras clínicas e gerar sugestões de diagnóstico.

Diagrama UML

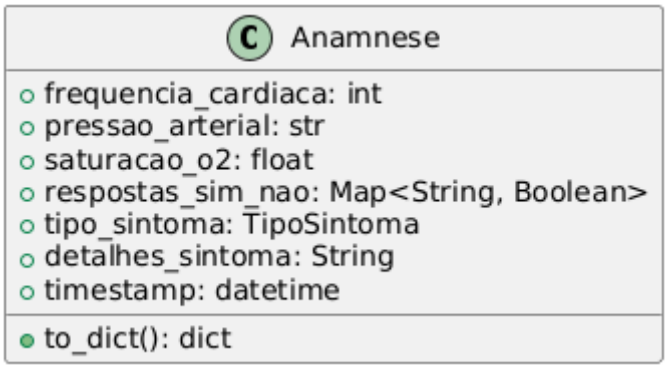


Fig. 8 – Diagrama UML da classe Anamnese

Classe Diagnostico

A classe **Diagnostico** tem como responsabilidade representar cada hipótese clínica gerada pelo módulo de decisão, carregando as informações necessárias para exibição e posterior registro no histórico do paciente:

Atributos

- categoria (TipoSintoma) # o grupo principal de sintomas
- descricao (str) # ex. “Apendicite”, “Pneumonia”
- exames_sugeridos (List[str]) # exames recomendados

Métodos

__str__(self) -> str: Retorna uma string formatada contendo categoria, descrição e, se houver, os exames sugeridos (por exemplo: "Gastrointestinal: Apendicite | Exames: Ultrassonografia abdominal").

Integração

Consulta: É instanciada dentro de Medico.sugerir_diagnosticos(), que percorre as regras clínicas, e cria um objeto Diagnostico para cada hipótese. Depois, retorna a lista final ao usuário.

Diagrama UML

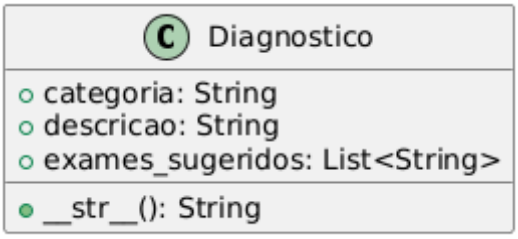


Fig. 9 – Diagrama UML da classe Diagnostico

Classe SistemaMediclass

A Classe SistemaMediclass, implementada no módulo sistema.py, controla a interface de linha de comando (CLI). Resumidamente, controla os processos de (a) login; (b) cadastro, busca e atualização de pacientes; (c) menu e acessos controlados a métodos; (d) execução e entrega dos métodos

Atributos

- usuarios: dict[str, Profissional] – mapeia logins para instâncias de profissionais cadastrados.
- pacientes: dict[str, Paciente] – mapeia CPFs para instâncias de pacientes registrados.

Métodos

registrar_usuario(self, usuario: Profissional) → None
Adiciona um objeto Profissional ao dicionário de usuários, usando seu login como chave.

login(self) → Profissional | None
Solicita login e senha via input(), autentica chamando Profissional.autenticar() e retorna o usuário ou None em caso de credenciais inválidas.

menu_principal(self, usuario: Profissional) → None
Exibe repetidamente um menu com opções enumeradas (registro de entrada, triagem,

consulta, visualização, exportação, adição de exame, logout) e despacha para os métodos escolhidos.

op_registrar_entrada(self, usuario: Profissional) → None
Pede CPF; se paciente não existir, solicita dados (nome, contato, convênio, data de nascimento, leito) e cria nova instância de Paciente; registra entrada no histórico.

op_triagem(self, usuario: Profissional) → None
Verifica se usuario é instância de Enfermeiro; em caso afirmativo, solicita CPF e chama Enfermeiro.triagem(paciente), caso contrário exibe “Acesso negado”.

op_diagnostico(self, usuario: Profissional) → None
Verifica se usuario é Medico; solicita CPF, chama Medico.sugerir_diagnosticos(paciente), exibe sugestões e pergunta ao médico se deseja solicitar exame, gerar receituário ou declaração.

op_visualizar_prontuario(self, usuario: Profissional) → None
Solicita CPF e exibe no terminal dados cadastrais, histórico médico e última anamnese do paciente.

op_exportar_prontuario(self, usuario: Profissional) → None
Semelhante a op_visualizar_prontuario, mas escreve todas as informações em um arquivo TXT formatado como prontuário.

op_adicionar_exame(self, usuario: Profissional) → None
Verifica se usuario é Tecnico; solicita CPF e chama Tecnico.adicionar_exame_sistema(paciente), caso contrário exibe “Acesso negado”.

executar(self) → None
Loop principal: chama login() e, enquanto este retornar um usuário válido, chama menu_principal(). Encerra quando login() retorna None.

Diagrama UML

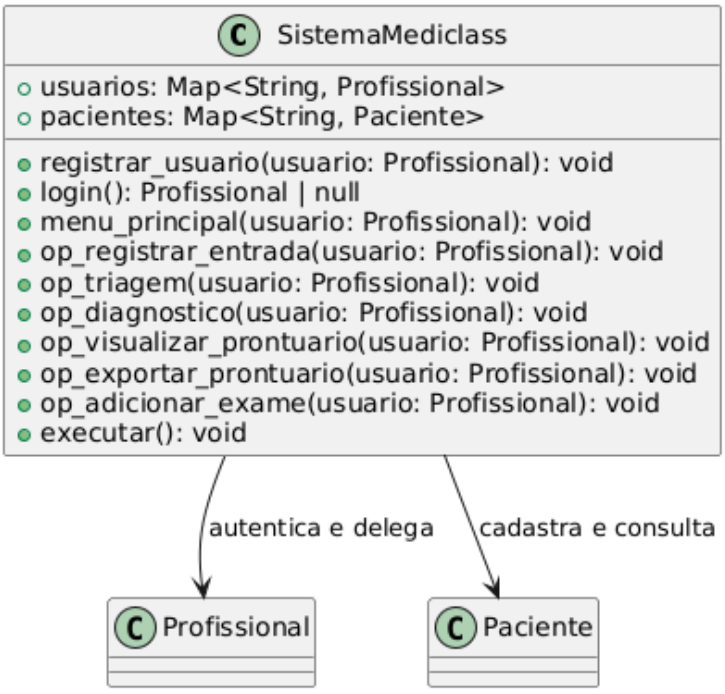


Fig. 10 – Diagrama UML da classe SistemaMediclass

A Programação Orientada a Objetos no MediClass

Abstração e Encapsulamento

Módulo Anamnese (anamnese.py) define a classe Anamnese, que encapsula tudo que é relevante à triagem (freq. cardíaca, pressão, oximetria, perguntas e respostas, tipo de sintoma, timestamp) e expõe apenas to_dict() para uso externo anamnese.

Classe Paciente (paciente.py) mantém seu histórico médico em arquivo privado (_historico_file) e fornece métodos públicos como registrar_entrada(), atualizar_historico() e consultar_historico() para manipulação segura desses dados paciente.

Classe Abstrata Profissional (profissionais.py) oculta sua senha real em _senha_hash e só permite autenticação via autenticar(), impedindo acesso direto ao atributo profissionais.

Herança

Superclasse Profissional reúne atributos e métodos comuns (nome, registro, login, hash de senha, contratos para cadastro/edição de pacientes) profissionais.

Subclasses:

Enfermeiro herda autenticação e adiciona triagem(paciente: Paciente) para coletar sinais vitais e gerar Anamnese.

Medico herda e adiciona sugerir_diagnosticos(paciente: Paciente) além de geração de receituário e declaração (produz objetos Diagnostico) .

Tecnico herda e implementa adicionar_exame_sistema(paciente: Paciente) para registrar resultados de exames profissionais.

Polimorfismo e Controle de Acesso

Na classe **SistemaMediclass** (sistema.py), o método genérico login() retorna sempre um Profissional, sem expor o tipo concreto sistema.

Em seguida, operações como op_triagem, op_diagnostico e op_adicionar_exame verificam dinamicamente o tipo com isinstance(...), mas poderiam simplesmente chamar métodos polimórficos sem conhecer a classe exata:

```
user = sistema.login()
# Polimorfismo: trata o retorno como Profissional,
# mas cada subclasse implementa seu próprio comportamento.
if hasattr(user, 'triagem'):
    user.triagem(paciente)       # Enf. executa triagem
elif hasattr(user, 'sugerir_diagnosticos'):
    user.sugerir_diagnosticos(paciente)  # Med. faz diagnóstico
```

Composição e Associação

SistemaMediclass agrega coleções de Profissional e Paciente, centralizando o fluxo de interação sistema.

Cada Paciente mantém referência a sua última Anamnese e a vários registros de exames, demonstrando composição de objetos de domínio.

Incrementos e futuras versões

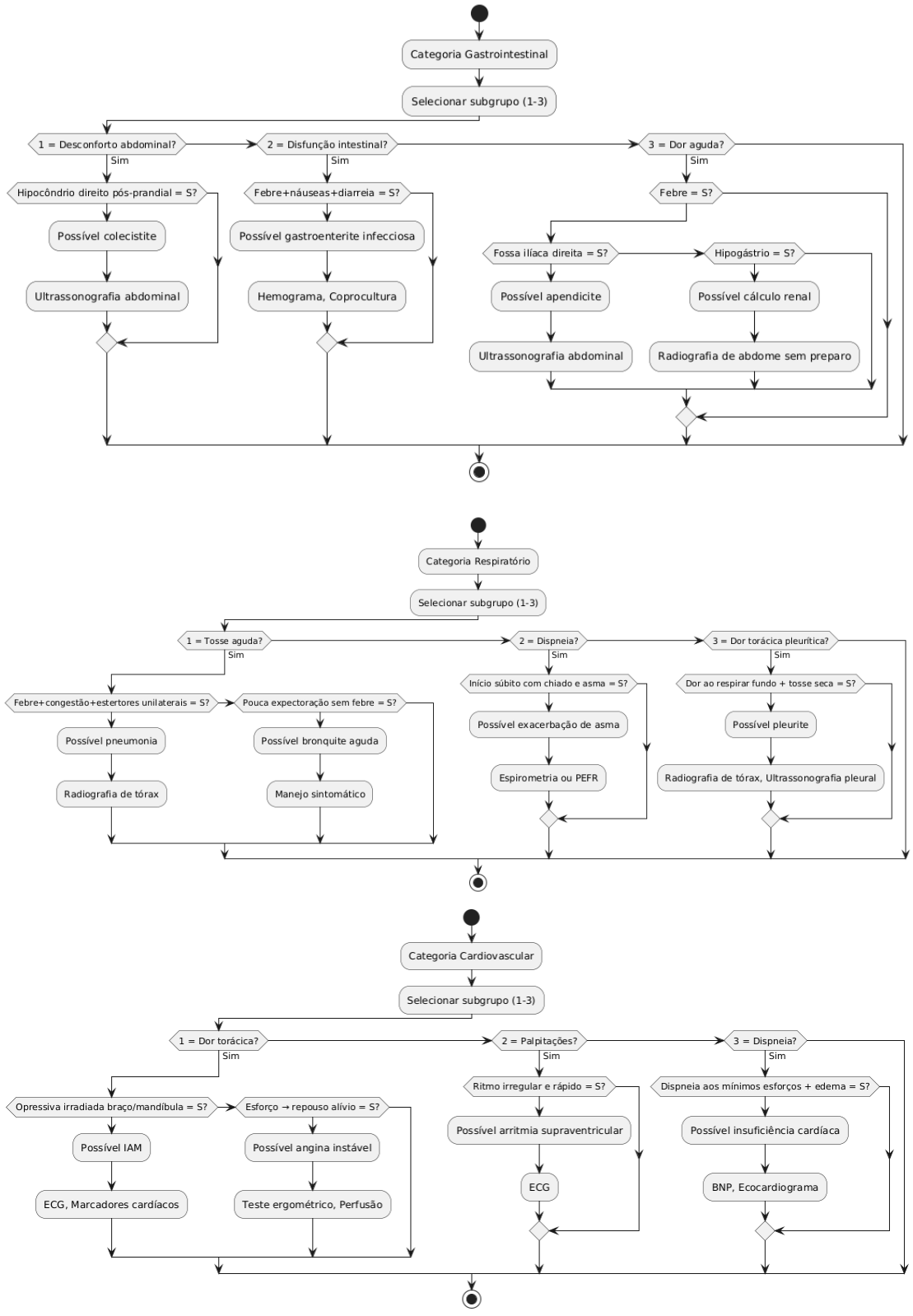
Além dos exemplos acima, como o Mediclass foi estruturado com classes bem definidas, aproveitando os pilares da POO, a implementação de novas funcionalidades é intuitiva.

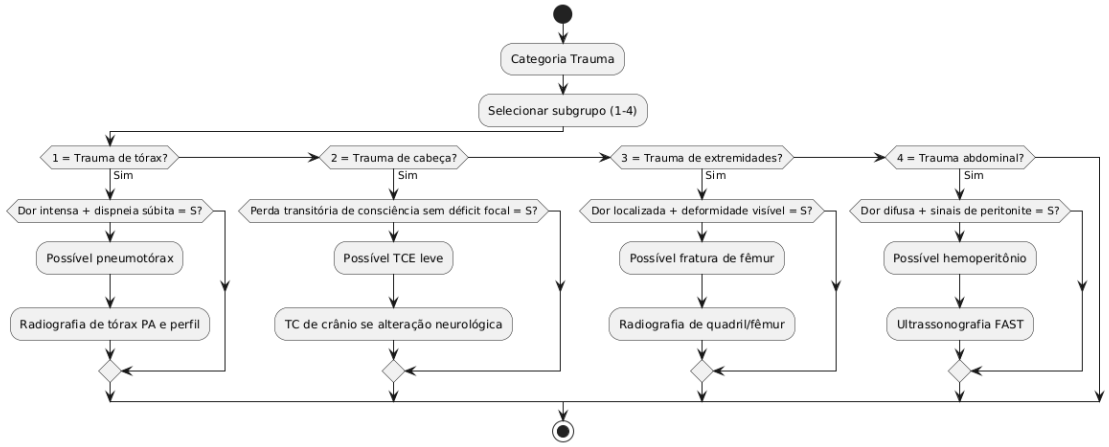
Uma extensão natural e de rápida implementação seria criar uma nova classe **Gerente**, também derivada de **Profissional**, que adiciona atributos, que podem ser privados (por exemplo, _contagem_consultas, _contagem_triagens e _contagem_exames) e métodos como registrar_consulta(), registrar_triagem() e registrar_exame(), para acompanhar a produtividade no hospital.

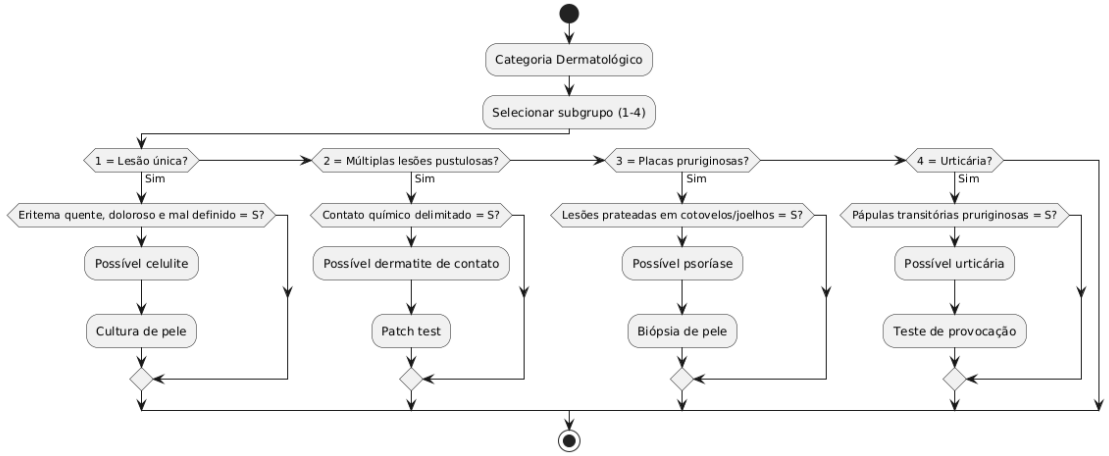
E assim, sempre que um médico, enfermeiro ou técnico executar sua operação (via op_diagnostico, op_triagem ou op_adicionar_exame no SistemaMediclass), o Gerente poderia incrementar automaticamente os respectivos contadores e disponibilizar relatórios para cada classe ou seu objeto.

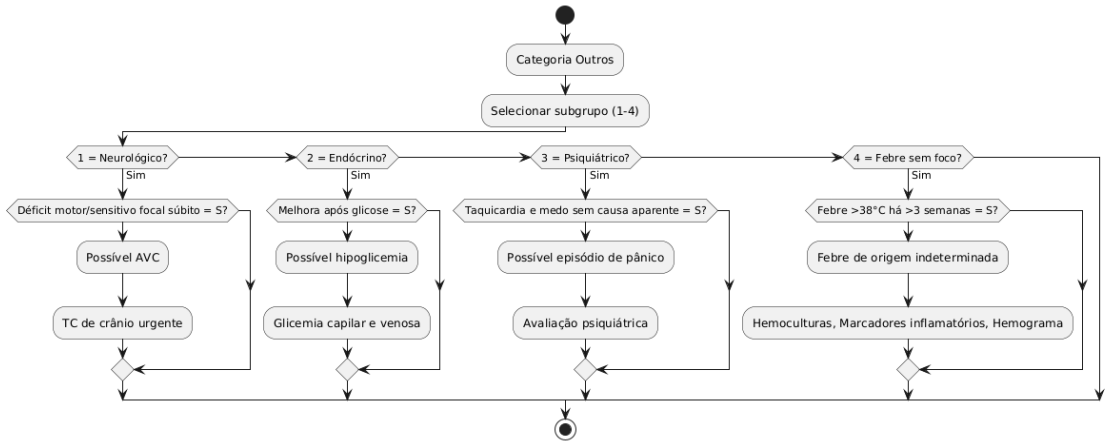
Apêndice A

Árvore de decisões do sistema de apoio ao diagnóstico, do método Medico.sugerir_diagnosticos().




Categoria Trauma
Selecionar subgrupo (1-4)
1 = Trauma de tórax?
2 = Trauma de cabeça?
3 = Trauma de extremidades?
4 = Trauma abdominal?
Sim
Dor intensa + dispneia súbita = S?
Possível pneumotórax
Radiografia de tórax PA e perfil
Perda transitória de consciência sem déficit focal = S?
Possível TCE leve
TC de crânio se alteração neurológica
Dor localizada + deformidade visível = S?
Possível fratura de fêmur
Radiografia de quadril/fêmur
Dor difusa + sinais de peritonite = S?
Possível hemoperitônio
Ultrassonografia FAST


Categoria Dermatológico
Selecionar subgrupo (1-4)
1 = Lesão única?
2 = Múltiplas lesões pustulosas?
3 = Placas pruriginosas?
4 = Urticária?
Sim
Eritema quente, doloroso e mal definido = S?
Possível celulite
Cultura de pele
Contato químico delimitado = S?
Possível dermatite de contato
Patch test
Lesões prateadas em cotovelos/joelhos = S?
Possível psoríase
Biópsia de pele
Pápulas transitórias pruriginosas = S?
Possível urticária
Teste de provocação


Categoria Outros
Selecionar subgrupo (1-4)
1 = Neurológico?
2 = Endócrino?
3 = Psiquiátrico?
4 = Febre sem foco?
Sim
Déficit motor/sensitivo focal súbito = S?
Possível AVC
TC de crânio urgente
Melhora após glicose = S?
Possível hipoglicemia
Glicemia capilar e venosa
Taquicardia e medo sem causa aparente = S?
Possível episódio de pânico
Avaliação psiquiátrica
Febre >38°C há >3 semanas = S?
Febre de origem indeterminada
Hemoculturas, Marcadores inflamatórios, Hemograma