Centro de Informática
UFPE

Ana Raquel Rodrigues da Silva
Izadora Taline Gonçalves de Andrade
Laura Virgínia do Nascimento Fonseca
Lais Saraiva Peixoto Costa
Nathan Barbosa dos Santos

# STUDIO YOGA - MANAGER

# (Projeto de Estrutura de Dados Orientada a Objetos)

Recife
2025

# 1. Introdução

Este documento apresenta o projeto "Studio Yoga", desenvolvido para a disciplina de Estruturas de Dados Orientadas a Objetos (CIN0135) do Centro de Informática (CIn) da UFPE.

O objetivo foi projetar e implementar um Sistema de Informação (Opção 1) funcional que aplicasse os conceitos fundamentais da Programação Orientada a Objetos (POO). O sistema escolhido gerencia as operações de um estúdio de yoga, permitindo o cadastro de praticantes e instrutores, o agendamento de aulas e a inscrição de alunos nessas aulas.

O sistema foi desenvolvido em C++, utiliza CMake para gerenciamento de build e arquivos de texto (.txt) para persistência de dados.

Utilizamos uma abordagem bottom-up para implementação do sistema, fazendo primeiro as classes e a configuração da persistência em arquivos para depois criarmos a UI.

# 2. Arquitetura do Sistema

A arquitetura do sistema é baseada no padrão Arquitetura em Camadas (Layered Architecture), uma variação do MVC (Model-View-Controller) adaptada para um aplicativo de console.

O pilar do design é a Separação de Responsabilidades (Separation of Concerns). Em vez de uma única classe monolítica, o código é organizado em camadas independentes:

- **Camada de Modelos (/models):** O "M" do MVC
    - O que faz: Define as entidades e estruturas de dados ("o que" é o sistema).
    - Exemplos: Pessoa.h, Praticante.h, Aula.h, HotYoga.h.
    - Trabalho: Guarda os dados e a lógica de negócios que pertence apenas àquele modelo (ex: aula->isLotada()).
- **Camada de Visão (View):** O "V" do MVC
    - O que faz: É o "Front-end" do console.
    - Exemplos: O método StudioManager::run(), o utils/Color.h e o gerarRelatorioHTML().
    - Trabalho: É responsável apenas por exibir informações (std::cout), aplicar cores e capturar a entrada (std::cin). Ele não toma decisões; ele apenas chama o Controlador.
- **Camada de Serviço (Controller):** O "C" do MVC
    - O que faz: É o "cérebro" da aplicação.
    - Exemplos: O services/StudioManager.h e seus métodos (ex: cadastrarPraticante(), atualizarPraticante()).
    - Trabalho: Coordena todas as outras camadas.
- **Camada de Dados (Persistência):**
    - O que faz: Lida com salvar e carregar do "banco de dados" (.dat).
    - Exemplos: data/DataManager.h
    - Trabalho: Esta camada é altamente desacoplada. O StudioManager (Controlador) não sabe *como* os dados são salvos; ele apenas diz: DataManager::salvarPraticantes(meus_vetores). Isso nos permite, no futuro, trocar os arquivos .dat por um banco de dados SQL real, mexendo apenas na camada `data/`.

```
StudioYoga-Manager-cpp/
|
+--- include/              # 1. Cabeçalhos (.h) - As "Camadas"
|   |
|   +--- models/           # --- Camada de Modelos (O que é?)
|   |   |--- Pessoa.h
|   |   |--- Praticante.h
|   |   |--- Instrutor.h
|   |   |--- Plano.h
|   |   |--- Aula.h
|   |   |--- HotYoga.h
|   |   \--- YogaPets.h
|   |
|   +--- services/         # --- Camada de Serviço (O Cérebro)
|   |   \--- StudioManager.h
|   |
|   +--- data/             # --- Camada de Dados (O Banco de Dados)
|   |   \--- DataManager.h
|   |
|   +--- validator/        # --- Camada de Validação
|   |   \--- Validator.h
|   |
|   \--- utils/            # --- Utilitários (O "Front-end" TUI)
|       \--- Color.h
```

```
|
+--- src/                # 2. Implementações (.cpp)
|   |
|   +--- models/
|   |   |--- Pessoa.cpp
|   |   |--- Praticante.cpp
|   |   |--- ... (etc.)
|   |
|   +--- services/
|   |   \--- StudioManager.cpp
|   |
|   +--- data/
|   |   \--- DataManager.cpp
|   |
|   \--- validator/
|       \--- Validator.cpp
|
+--- build/              # 3. Arquivos Gerados (Onde o programa roda)
|   |--- StudioYogaManager.exe  # <<< PROGRAMA FINAL
|   |--- praticantes.dat       # <<< BANCO DE DADOS
|   |--- instrutores.dat
|   |--- planos.dat
|   \--- relatorio.html        # <<< RELATÓRIO HTML
|
```

```
+--- docs/                # 4. Documentação
|   |--- index.html
|   \--- style.css
|
|--- CMakeLists.txt        # A "Receita" do Projeto
|
|--- main.cpp              # Ponto de entrada (só chama o StudioManager)
|
\--- README.md             # O "Abstract" do projeto
```

# 3. Conceitos de POO Aplicados

O sistema foi modelado com base nos pilares da POO, aplicados dentro da arquitetura em camadas:

**Classes e Objetos:** O domínio do problema foi modelado em classes claras. As entidades principais são Pessoa, Praticante, Instrutor, Plano e Aula. A lógica de negócios é gerenciada pela classe StudioManager e a persistência pelo namespace DataManager.

**Encapsulamento:** As camadas escondem sua complexidade interna.

- **Nos Modelos:** Atributos (Praticante::idPlano, Pessoa::nome) são private, acessados por métodos públicos (Getters/Setters).
- **Na Arquitetura:** O StudioManager não sabe como o DataManager funciona (ele não vê fstream ou stringstream). A complexidade da persistência está encapsulada dentro da camada de dados.

**Herança:** Utilizada para reutilização de código e estabelecimento de relações "É UM TIPO DE":

- Praticante e Instrutor herdam da classe base Pessoa, compartilhando os atributos id, nome e email.
- HotYoga, YogaPets e YogaFlow herdam da classe base abstrata Aula,

compartilhando atributos como id, horario e limiteAlunos.

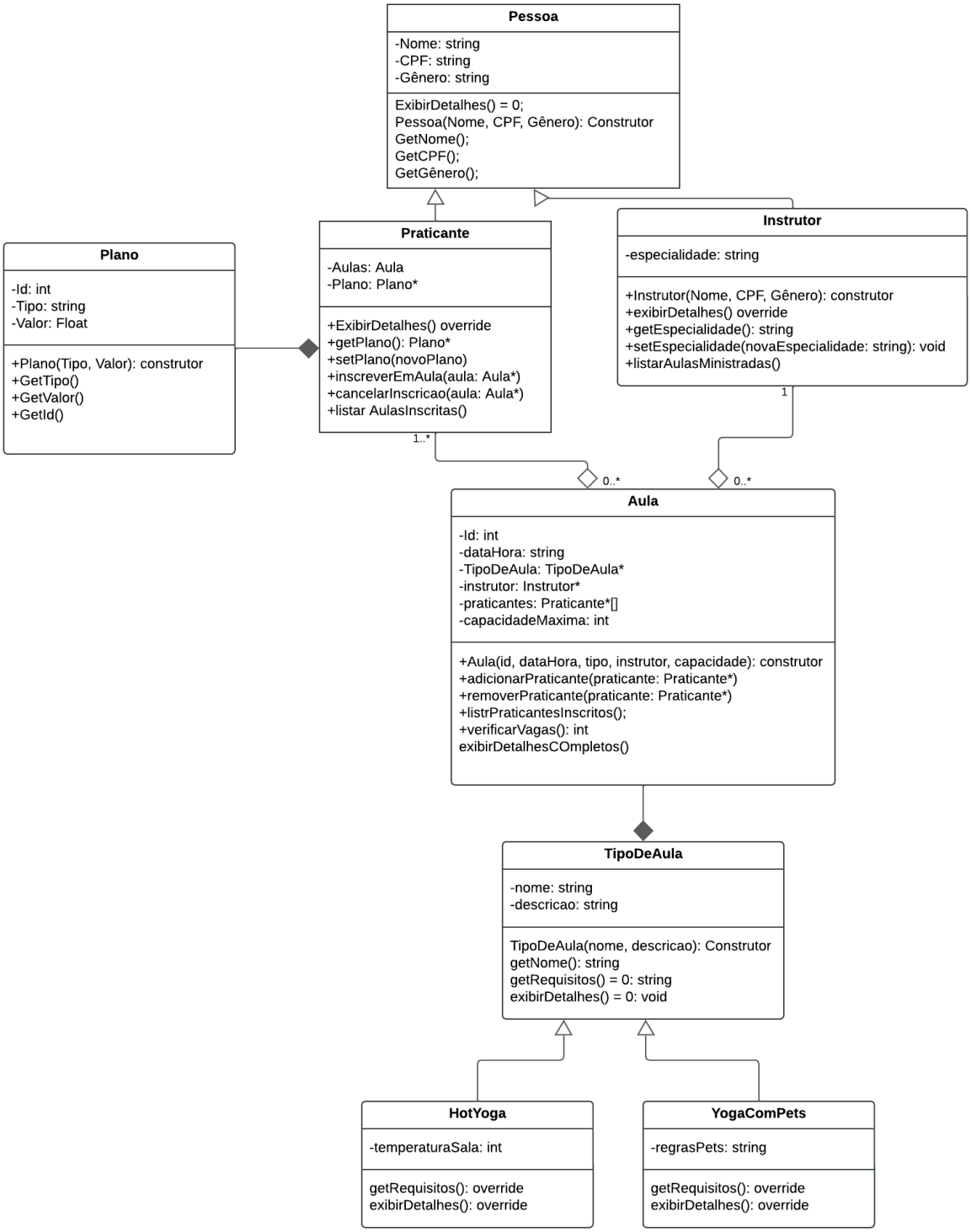
**Polimorfismo (em Tempo de Execução):** Este é o conceito mais avançado aplicado e é a chave para o gerenciamento de aulas.

- **Armazenamento:** As diferentes aulas (HotYoga, YogaPets, YogaFlow) são armazenadas em um único vetor da classe base: std::vector<Aula*> aulas. Isso só é possível com ponteiros (ou referências).
- **Métodos Virtuais:** O método exibirDetalhes() é virtual na classe Aula e override nas classes filhas. Ao listar as aulas, o StudioManager chama aulaPtr->exibirDetalhes(). O sistema decide em tempo real qual versão do método chamar (a de HotYoga, YogaPets ou YogaFlow), exibindo os detalhes corretos (como Temperatura) para cada tipo de aula.

**Padrões de Projeto:** O projeto implementa dois padrões de projeto clássicos:

- **Singleton (Estático):** Os namespaces DataManager e Validator agem como Singletons. Eles não podem ser instanciados e fornecem um ponto de acesso global e único para seus serviços (salvar/carregar e validar).
- **Simple Factory (Fábrica Simples):** O método StudioManager::cadastrarAula() age como uma "Fábrica". Baseado na escolha do usuário, ele constrói o objeto complexo correto (new HotYoga(...), new YogaPets(...) ou new YogaFlow(...)), mas retorna a interface simples da classe base (Aula*).

# 4. Diagrama de Classes (UML)



# 5. Divisão de Tarefas

Ana Raquel Rodrigues da Silva:

- Classes: HotYoga.h e Plano.h
- Arquitetura do projeto
- Implementação de UTI
- Vídeo
- Revisão de código

Izadora Taline Gonçalves de Andrade:

- Classes: Pessoa.h e Praticante.h
- Estruturação do crud
- Vídeo
- Revisão de código

Laura Virgínia do Nascimento Fonseca:

- Classes: Instrutor.h e YogaPets.h
- README.md
- Documentação
- Revisão de código

Lais Saraiva Peixoto Costa:

- Classes: TipoDeAula.h e YogaFlow.h
- Arquitetura do projeto
- Vídeo
- Revisão do código

Nathan Barbosa dos Santos:

- Classes: Aula.h
- Responsável pela arquitetura do projeto (CMake, estrutura de pastas, princípio da responsabilidade única ).
- Implementação da integração com o banco de dados (SQLite, DatabaseManager).
- Implementação da camada AulaRepository e da UI (main.cpp)