RELATÓRIO FINAL DO PROJETO: Sistema de Gestão FitLife (Projeto POO)

Disciplina: Programação Orientada a Objetos

Data: 23 de Novembro de 2025

Membros da Equipe

Pedro Henrique Rodrigues Jacques Pinheiro

Gabriel do Rego Lima Menezes

Henrique França de Souza Medeiros Maranguape

José Airton Rodrigues Galdino Júnior

Marina de Lima Fonseca

1. Contribuição Individual (O que o aluno fez no projeto)

Objetivo: Descrever as tarefas específicas, módulos ou classes desenvolvidas por cada membro da equipe.

1.1. Pedro Henrique Rodrigues Jacques Pinheiro (Configuração Central)

- Desenvolveu a classe central ServicoDeGestaoFitLife.java, responsável por orquestrar a lógica de todos os módulos.
- Implementou o CRUD completo de Modalidades, Professores e o agendamento de Aulas.
- Criou a lógica de persistência de dados em arquivos CSV para as entidades principais, substituindo a necessidade de banco de dados complexo.
- Implementou a lógica de diferenciação de horários, permitindo o cadastro de aulas e vinculação com professores.

1.2. Gabriel do Rego Lima Menezes & Marina de Lima Fonseca (Recursos e Relatórios de Desempenho)

Nota: Devido à complexidade do módulo, estes membros atuaram em conjunto (Pair Programming).

- Gabriel: Focou na estruturação da classe Frequencia.java, responsável por registrar a presença dos alunos (simulando o histórico de desempenho). Trabalhou na lógica de "Lookup" para garantir que frequências fossem ligadas a alunos existentes.

- Marina: Liderou a implementação da biblioteca externa iText. Desenvolveu a classe GeradorDeRelatorio.java, responsável por converter os dados em memória (Listas) para um documento PDF formatado e profissional.

- Ambos trabalharam na refatoração do código para suportar a geração de relatórios unificados, integrando dados de todos os outros módulos (Financeiro, Gestão e Alunos) no documento final.

1.3. Henrique França de Souza Medeiros Maranguape (Financeiro)

- Desenvolveu o ServicoFinanceiro.java e a classe Transacao.java.

- * Implementou a lógica de fluxo de caixa, permitindo registrar pagamentos e validar se um aluno existe antes de cobrar (Injeção de dependência do Serviço de Gestão).

- * Criou o RelatorioFinanceiro.java, utilizando Java Streams API para realizar cálculos complexos, como somatória de receita por tipo de plano e estimativa de lucro por modalidade.

- * Implementou a persistência separada para o arquivo transacoes.csv.

1.4. José Airton Rodrigues Galdino Júnior (Planos e Alunos)

- Implementou a hierarquia de classes de planos (Plano, PlanoBasico, PlanoVip, PlanoMensal), aplicando herança e polimorfismo.

- Desenvolveu a classe Aluno.java e a lógica de validação de idade (regra de negócio para menores de 18 anos).

- Responsável pela lógica de regras de acesso (método temAcessoExclusivoAulas), garantindo que a regra de negócio VIP fosse respeitada.

- Auxiliou na integração entre o cadastro de alunos e a atribuição de planos no momento da matrícula.

2. Funcionalidades Implementadas

Objetivo: Listar as funcionalidades que o grupo conseguiu entregar, com ênfase na aplicação dos conceitos da disciplina (ex: POO).

Funcionalidades de Negócio

- Gestão de Aulas e Professores: Cadastro completo, edição e remoção de professores e modalidades, com persistência em disco.

- Polimorfismo em Planos: O sistema distingue automaticamente privilégios (acesso a aulas VIP e reservas) dependendo se a instância do plano é PlanoVip ou PlanoBasico.

- Controle Financeiro Integrado: Registro de pagamentos com validação de integridade (só recebe de alunos cadastrados) e relatórios de receita.

- Geração de Relatórios em PDF: Exportação de dados gerenciais completos (Alunos, Professores, Modalidades) para arquivo PDF utilizando biblioteca de terceiros.

Aplicação de Conceitos Técnicos:

- Encapsulamento: Todos os atributos das classes (Aluno, Aula, Transacao) são privados e acessados via Getters/Setters.

- Abstração e Herança: Uso da classe abstrata Plano como base para especializações (PlanoAnual, PlanoVip), facilitando a manutenção.

- Persistência em Arquivos (CSV): Substituição do banco de dados SQL por manipulação de arquivos de texto (BufferedReader/BufferedWriter), garantindo que os dados não sejam perdidos ao fechar o programa.

- Java Streams: Utilizado massivamente no módulo financeiro para filtrar, mapear e somar valores de transações de forma funcional e limpa.

3. Funcionalidades Não Implementadas

Objetivo: Listar as funcionalidades planejadas que não foram concluídas e justificar o motivo.

1. **Banco de Dados Relacional (SQL):**

O projeto utiliza arquivos CSV em vez de um banco MySQL/PostgreSQL.

Motivo: Devido ao prazo curto e à complexidade de configuração de ambiente para todos os membros, optou-se por persistência em arquivos locais, que atende ao requisito de manter os dados salvos.

2. **Interface Gráfica (GUI):**

O sistema roda inteiramente no Console (Terminal).

Motivo: O foco do trabalho foi a lógica de Back-End e regras de POO. O tempo foi investido na robustez das classes de serviço em vez de telas visuais.

3. **Controle Granular de Equipamentos:**

O sistema gerencia "Modalidades", mas não o estoque individual de halteres ou esteiras.

Motivo: Simplificação do escopo para focar na gestão de pessoas e financeiro, que são o core business da academia.

4. Experiência do Aluno

1.1. Pedro Henrique Rodrigues Jacques Pinheiro

O que mais gostou

Ver a integração dos módulos funcionando na classe Main, onde o Financeiro "conversa" com a Gestão.

Maiores dificuldades

Tratar as exceções na leitura de arquivos CSV (erros de formatação e arquivos inexistentes).(OBS:NEM NOÉ)

O quanto prendeu

Aprendeu muito sobre modularização, dependências e boas práticas de organização através do Maven. Reforçou habilidades de implementação de classes, entendimento de responsabilidades, A importância de separar as classes de modelo (Dados) das classes de serviço (Lógica).

1.2. Gabriel do Rego Lima Menezes

O que mais gostou

Trabalhar na parte de configuração e organização do repositório trouxe a sensação de estar ajudando o sistema a ficar mais profissional e padronizado e ver resultado final do PDF gerado. Ver os dados saindo do console e indo para um documento real foi gratificante.

Maiores dificuldades

Configurar a biblioteca externa (itextpdf.jar) no projeto e lidar com conflitos de dependência no IntelliJ.

O quanto aprendeu

Organização de arquivos e padronização do ambiente de desenvolvimento. Também desenvolveu melhor entendimento sobre como gerenciar versões, branches e limpeza do repositório de forma adequada, Manipulação de bibliotecas de terceiros e formatação de documentos via código.

1.3. Henrique França de Souza Medeiros Maranguape

O que mais gostou

Criar os cálculos de receita usando Java Streams. É muito mais limpo do que usar laços for tradicionais.

Maiores dificuldades

A maior dificuldade foi lidar com a remoção dos arquivos antigos e garantir que nada importante fosse perdido no processo. Lidar com formatos de data (String vs LocalDate) na hora de salvar e ler do arquivo CSV.

O quanto aprendeu

Conceitos de Business Intelligence (BI) aplicados a objetos Java e persistência de transações. Além disso, ganhou mais segurança ao lidar com commits iniciais e na preparação de um ambiente limpo e funcional para toda a equipe.

1.4. José Airton Rodrigues Galdino Júnior

O que mais gostou

Implementar a lógica VIP. Foi interessante ver como uma simples mudança no ID do plano alterava o comportamento do aluno no sistema.

Maiores dificuldades

Sincronizar os IDs dos planos no CSV com a lógica do código Java.(OBS:NEM NOÉ)

O quanto aprendeu

Como utilizar polimorfismo na prática para evitar múltiplos if/else no código. Como ser mais paciente e organizado com os coleguinhas

1.5. Marina de Lima Fonseca

O Que mais gostou

Gostou de participar diretamente da construção da ideia inicial do sistema, ajudando a dar forma ao projeto desde o começo. Também apreciou trabalhar na organização do código-fonte e na criação das classes base, vendo o projeto ganhar estrutura e clareza. A parte de documentação — como escrever o README e explicar decisões técnicas — também foi algo que trouxe satisfação por deixar o projeto mais compreensível para todos.

Maiores dificuldades

A maior dificuldade foi manter a organização do código enquanto o sistema ainda estava em desenvolvimento inicial, especialmente ao revisar estrutura de pastas, nomeação de classes e divisão lógica entre módulos. Também foi desafiador produzir documentação clara e completa, como o README e partes do relatório final, exigindo atenção aos detalhes e cuidado para explicar bem o funcionamento do projeto. Por fim, o fato de participar de um grupo já estruturado e buscar contribuir da mesmo forma com o encaminhamento.

O quanto aprendeu

Aprendeu bastante sobre estruturação de projetos desde o início, reforçando boas práticas de organização de código e criação de classes base. Aprendeu também sobre documentação técnica, entendendo melhor como explicar arquitetura, fluxo do sistema e decisões de design. Além disso, consolidou conhecimentos sobre padronização em projetos

Maven e sobre como deixar um repositório mais apresentável e funcional para outros desenvolvedores.

5. Bibliotecas Utilizadas e Referências

5.1. Bibliotecas e Tecnologias

Tecnologia	Versão	Motivo da Escolha
Java JDK 17+:	[Ex: 17]	Linguagem base
iText PDF (v5.5.13.4)	(v5.5.13.4)	Biblioteca externa utilizada para a geração dos relatórios gerenciais em PDF. Escolhida pela estabilidade e facilidade de uso em comparação à versão 7.
Arquivos CSV	N/A	Para manipulação de arquivos CSV (BufferedReader, BufferedWriter).

5.2. Referências Consultadas

- Documentação Oficial do Java (Oracle) - Streams API.
- iText 5 Examples - "Creating a simple PDF".
- Conteúdo de aula: "Abstração e Polimorfismo em Java".

6. Impressão de Classe Importante

Classe Escolhida: `ServicoDeGestaoFitLife.java`

```
package com.fitlife;

import com.fitlife.Aluno.Aluno;
import com.fitlife.Aula.Aula;
import com.fitlife.Modalidade.Modalidade;
import com.fitlife.Plano.Plano;
import com.fitlife.Professor.Professor;

import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
```

```java
// O Cérebro do Sistema. Se essa classe cair, a academia fecha. Aconteça o que acontecer não faça merda aqui (GABRIEL).
public class ServicoDeGestaoFitLife {

    // Listas em memória (Nosso banco de dados é a memória RAM, torça para não faltar luz)
    private List<Professor> professores = new ArrayList<>();
    private List<Modalidade> modalidades = new ArrayList<>();
    private List<Aula> aulas = new ArrayList<>();
    private List<Aluno> alunos = new ArrayList<>();
    private List<?> frequencias = new ArrayList<>();

    // Arquivos CSV onde a mágica persiste
    private static final String PROFESSOR_ARQUIVO = "professores.csv";
    private static final String MODALIDADE_ARQUIVO = "modalidades.csv";
    private static final String AULA_ARQUIVO = "aulas.csv";
    private static final String ALUNO_ARQUIVO = "alunos.csv";
    private static final String FREQUENCIA_ARQUIVO = "frequencias.csv";

    public ServicoDeGestaoFitLife() {
        carregarTodosDados(); //
    }

    // --- MÉTODOS DE BUSCA  ---

    public void adicionarAlunoParaTeste(Aluno aluno) {
        this.alunos.add(aluno);
    }

    public Optional<Professor> buscarProfessorPorId(int id) {
        return professores.stream().filter(p -> p.getId() == id).findFirst();
    }

    public Optional<Modalidade> buscarModalidadePorId(int id) {
        return modalidades.stream().filter(m -> m.getId() == id).findFirst();
    }

    public Optional<Aluno> buscarAlunoPorId(long id) {
        return alunos.stream().filter(a -> a.getId() == id).findFirst();
    }

    // --- PERSISTÊNCIA ---

    private void carregarTodosDados() {
        carregarDadosSimples(MODALIDADE_ARQUIVO, modalidades, Modalidade.class);
        carregarDadosSimples(PROFESSOR_ARQUIVO, professores, Professor.class);
        carregarDadosSimples(ALUNO_ARQUIVO, alunos, Aluno.class);
        carregarAulas(AULA_ARQUIVO);
    }
```

```java
// Método genérico para ler qualquer CSV simples.
private <T> void carregarDadosSimples(String nomeArquivo, List<T> lista, Class<T> classe) {
    File arquivo = new File(nomeArquivo);
    if (!arquivo.exists()) { return; }

    try (BufferedReader br = new BufferedReader(new FileReader(arquivo))) {
        String linha;
        while ((linha = br.readLine()) != null) {
            if (!linha.trim().isEmpty()) {
                try {
                    T objeto = classe.getConstructor(String.class).newInstance(linha);
                    lista.add(objeto);
                } catch (Exception e) {
                    System.err.println("Linha corrompida no arquivo " + nomeArquivo + ". Ignorando...");
                }
            }
        }
    } catch (IOException e) {
        System.err.println("Erro de I/O: " + e.getMessage());
    }
}

// Carrega aulas
private void carregarAulas(String nomeArquivo) {
    File arquivo = new File(nomeArquivo);
    if (!arquivo.exists()) return;

    try (BufferedReader br = new BufferedReader(new FileReader(arquivo))) {
        String linha;
        while ((linha = br.readLine()) != null) {
            if (!linha.trim().isEmpty()) {
                try {
                    String[] dados = linha.split(";");
                    if (dados.length != 6) throw new IllegalArgumentException("CSV de Aula inválido.");

                    int idAula = Integer.parseInt(dados[0].trim());
                    int idModalidade = Integer.parseInt(dados[1].trim());
                    int idProfessor = Integer.parseInt(dados[2].trim());
                    String horario = dados[3].trim();
                    String dia = dados[4].trim();
                    boolean isVIP = Boolean.parseBoolean(dados[5].trim());

                    // Reconstrói os objetos (Lookups)
                    Optional<Modalidade> mOpt = buscarModalidadePorId(idModalidade);
                    Optional<Professor> pOpt = buscarProfessorPorId(idProfessor);

                    if (mOpt.isPresent() && pOpt.isPresent()) {
                        aulas.add(new Aula(idAula, mOpt.get(), pOpt.get(), horario, dia, isVIP));
                    }
                } catch (Exception e) {
```

```
                    System.err.println("Erro ao ler aula: " + e.getMessage());
                }
            }
        }
    } catch (IOException e) {
        System.err.println("Erro de leitura: " + e.getMessage());
    }
}

// Salva tudo de volta no disco
public void salvarTodosDados() {
    salvarEntidades(MODALIDADE_ARQUIVO, modalidades);
    salvarEntidades(PROFESSOR_ARQUIVO, professores);
    salvarEntidades(AULA_ARQUIVO, aulas);
    salvarEntidades(ALUNO_ARQUIVO, alunos);
    salvarEntidades(FREQUENCIA_ARQUIVO, frequencias);
}

private void salvarEntidades(String nomeArquivo, List<?> lista) {
    if (lista.isEmpty()) { new File(nomeArquivo).delete(); return; }

    try (BufferedWriter bw = new BufferedWriter(new FileWriter(nomeArquivo))) {
        for (Object item : lista) {
            String csvLine = "";
            // Polimorfismo manual feio, mas funciona
            if (item instanceof Professor) csvLine = ((Professor) item).toCSV();
            else if (item instanceof Modalidade) csvLine = ((Modalidade) item).toCSV();
            else if (item instanceof Aula) csvLine = ((Aula) item).toCSV();
            else if (item instanceof Aluno) csvLine = ((Aluno) item).toCSV();
            else continue;

            bw.write(csvLine);
            bw.newLine();
        }
    } catch (IOException e) {
        System.err.println("Erro ao salvar: " + e.getMessage());
    }
}

// --- REGRAS DE NEGÓCIO (CRUD) ---

public Professor cadastrarProfessor(String nome, String registro, String especializacao) throws
IllegalArgumentException {
    int novoId = professores.stream().mapToInt(Professor::getId).max().orElse(0) + 1;
    Professor novoProfessor = new Professor(novoId, nome, registro, especializacao);
    professores.add(novoProfessor);
    salvarTodosDados();
    return novoProfessor;
}
```

```
    public Modalidade cadastrarModalidade(String nome, String descricao) throws
IllegalArgumentException {
        int novoId = modalidades.stream().mapToInt(Modalidade::getId).max().orElse(0) + 1;
        Modalidade novaModalidade = new Modalidade(novoId, nome, descricao);
        modalidades.add(novaModalidade);
        salvarTodosDados();
        return novaModalidade;
    }

    public Aula agendarNovaAula(int modalidadeId, int professorId, String horario, String dia, boolean
isVIP) throws Exception {
        Modalidade modalidade = buscarModalidadePorId(modalidadeId).orElseThrow(() -> new
Exception("Modalidade sumiu!"));
        Professor professor = buscarProfessorPorId(professorId).orElseThrow(() -> new
Exception("Professor sumiu!"));

        int novoId = aulas.stream().mapToInt(Aula::getId).max().orElse(0) + 1;
        Aula novaAula = new Aula(novoId, modalidade, professor, horario, dia, isVIP);
        aulas.add(novaAula);
        salvarTodosDados();
        return novaAula;
    }

    // Cadastro de Aluno com a Regra de Negócio de Menor de Idade
    public Aluno cadastrarNovoAluno(String nome, int idade, String autorizacaoStatus, Plano planoInicial)
throws IllegalArgumentException {
        if (idade < 18 && !"SIM".equalsIgnoreCase(autorizacaoStatus)) {
            throw new IllegalArgumentException("Sem autorização dos pais, sem treino. Regras são regras.");
        }

        long novoAlunoId = alunos.stream().mapToLong(Aluno::getId).max().orElse(0L) + 1;
        Aluno novoAluno = new Aluno(novoAlunoId, nome, idade, planoInicial);
        this.alunos.add(novoAluno);
        salvarTodosDados();
        return novoAluno;
    }

    // --- FILTRO VIP ---
    public List<Aula> listarAulasDisponiveis(long alunoId) {
        Aluno aluno = buscarAlunoPorId(alunoId).orElse(null);
        if (aluno == null) return aulas.stream().filter(aula ->
!aula.isExclusivaVIP()).collect(Collectors.toList());

        return aulas.stream()
                .filter(aula -> {
                    if (!aula.isExclusivaVIP()) return true; // Aula comum, entra todo mundo
                    // Se for VIP, pergunta pro Plano se pode entrar (Polimorfismo!)
                    return aluno.getPlano() != null && aluno.getPlano().temAcessoExclusivoAulas();
                })
                .collect(Collectors.toList());
```

```
    }

    // --- EDIÇÃO E REMOÇÃO ---

    public boolean removerAluno(long id) {
        boolean removeu = alunos.removeIf(a -> a.getId() == id);
        if (removeu) salvarTodosDados();
        return removeu;
    }

    public void editarAluno(long id, String novoNome, int novaIdade) throws Exception {
        Aluno aluno = buscarAlunoPorId(id).orElseThrow(() -> new Exception("Aluno fantasma? Não
achei."));
        if (novoNome != null && !novoNome.trim().isEmpty()) aluno.setNome(novoNome);
        if (novaIdade > 0) aluno.setIdade(novaIdade);
        salvarTodosDados();
    }

    public boolean removerProfessor(int id) {
        boolean removeu = professores.removeIf(p -> p.getId() == id);
        if (removeu) salvarTodosDados();
        return removeu;
    }

    public void editarProfessor(int id, String novoNome, String novaEsp) throws Exception {
        Professor p = buscarProfessorPorId(id).orElseThrow(() -> new Exception("Professor não
encontrado"));
        if (!novoNome.trim().isEmpty()) p.setNome(novoNome);
        if (!novaEsp.trim().isEmpty()) p.setEspecialidade(novaEsp);
        salvarTodosDados();
    }

    public boolean removerModalidade(int id) {
        boolean removeu = modalidades.removeIf(m -> m.getId() == id);
        if (removeu) salvarTodosDados();
        return removeu;
    }

    public void editarModalidade(int id, String novoNome, String novaDesc) throws Exception {
        Modalidade m = buscarModalidadePorId(id).orElseThrow(() -> new Exception("Modalidade não
encontrada"));
        if (!novoNome.trim().isEmpty()) m.setNome(novoNome);
        if (!novaDesc.trim().isEmpty()) m.setDescricao(novaDesc);
        salvarTodosDados();
    }

    // Getters para a galera (Cópia defensiva para ninguém estragar a lista original)
    public List<Modalidade> getTodasModalidades() { return new ArrayList<>(modalidades); }
    public List<Professor> getTodosProfessores() { return new ArrayList<>(professores); }
    public List<Aula> getTodasAulas() { return new ArrayList<>(aulas); }
```

```
    public List<Aluno> getTodosAlunos() { return new ArrayList<>(alunos); }
}
```

Justificativa: A classe `ServicoDeGestaoFitLife` é essencial porque centraliza toda a lógica do sistema, gerenciando professores, aulas, modalidades e alunos. Ela substitui o banco de dados usando arquivos CSV, aplica regras de negócio importantes, integra todas as entidades do projeto e organiza o fluxo da aplicação. É o núcleo funcional do FitLife e demonstra claramente conceitos fundamentais de POO.Esta classe é o coração do sistema, gerenciando todas as listas em memória e a persistência em disco