UNIVERSIDADE FEDERAL DE UBERLÂNDIA

CURSO: SISTEMAS DE INFORMAÇÃO

Alunos:

Alexandre Guersoni

Angélica Cordeiro

Caio Moreira

Guilherme Marcondes

Kaique Almeida

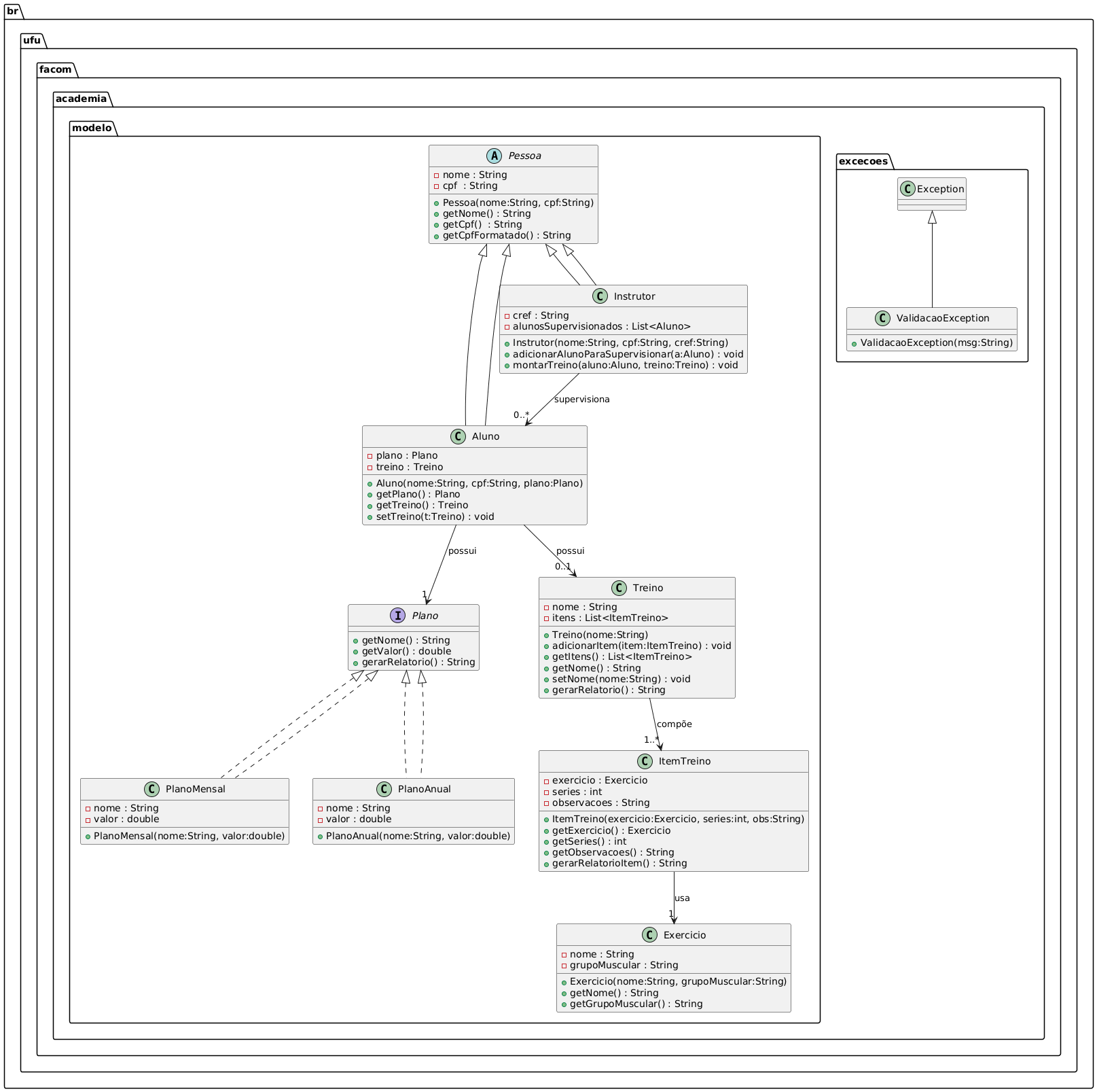
**SISTEMA DE ACADEMIA – RELATÓRIO TÉCNICO**

Uberlândia, 10 de setembro de 2025

**Introdução**

Este relatório documenta o desenvolvimento de um sistema de academia, modelado e implementado em Java, aplicando os pilares de POO (abstração, encapsulamento, herança e polimorfismo), uso de interfaces e tratamento de exceções.  
A arquitetura foi organizada em pacotes: modelo, exceções e main, destacando as entidades **Pessoa**, **Aluno**, **Instrutor**, **Plano** (interface com as implementações **PlanoMensal** e **PlanoAnual**), **Exercicio**, **ItemTreino** e **Treino**. Uma exceção verificada (**ValidacaoException**) centraliza validações de domínio (ex.: CPF e valores de planos).  
A aplicação é executada via console (classe Principal), com cadastros e geração de relatório textual do aluno.

**Diagrama UML de classes (PlantUML pronto)**



O diagrama representa a organização e os relacionamentos entre os principais elementos do projeto:

Pessoa é uma classe abstrata que serve como base para Aluno e Instrutor, reutilizando atributos comuns (nome e CPF).

Aluno possui um Plano (mensal ou anual, implementados a partir da interface Plano) e opcionalmente um Treino.

Instrutor supervisiona vários alunos e pode montar treinos para eles.

Treino é composto por vários ItemTreino, e cada item faz referência a um Exercicio específico, indicando séries e observações.

A interface Plano garante polimorfismo: permite que o sistema trate planos mensais e anuais de forma uniforme.

A exceção personalizada ValidacaoException centraliza regras de negócio, como validações de CPF e valores.

Esse desenho deixa claro os pilares de herança, polimorfismo, encapsulamento e composição, aplicados de maneira prática para representar o domínio de uma academia

**Demonstração dos conceitos de POO (com trechos do seu projeto)**

**1) Abstração + Herança (classe base Pessoa)**

As classes concretas Aluno e Instrutor herdam de Pessoa, reutilizando validações e atributos comuns.

public abstract class Pessoa {

private final String nome;

private final String cpf; // 11 dígitos

public Pessoa(String nome, String cpf) throws ValidacaoException {

if (nome == null || nome.isBlank()) throw new ValidacaoException("O nome não pode ser vazio.");

if (!cpfValido(cpf)) throw new ValidacaoException("CPF inválido.");

this.nome = nome.trim();

this.cpf = cpf.replaceAll("[^0-9]", "");

}

// getters...

}

**2) Encapsulamento (atributos privados + getters/setters)**

public class Aluno extends Pessoa {

private Plano plano;

private Treino treino;

public Aluno(String nome, String cpf, Plano plano) throws ValidacaoException {

super(nome, cpf);

this.plano = plano;

}

public Plano getPlano() { return plano; }

public Treino getTreino() { return treino; }

public void setTreino(Treino t) { this.treino = t; }

}

**3) Interface + Polimorfismo (Plano → Mensal | Anual)**

O código cliente trata Plano abstratamente, permitindo trocar a implementação sem alterar o restante.

public interface Plano {

String getNome();

double getValor();

String gerarRelatorio();

}

// Uso polimórfico

Plano plano = new PlanoMensal("Plano Básico", 99.90);

// ... ou ...

plano = new PlanoAnual("Plano Completo Anual", 1200.00);

**4) Composição e responsabilidade (Treino, ItemTreino, Exercicio)**

Treino t = new Treino("Treino A");

t.adicionarItem(new ItemTreino(new Exercicio("Supino Reto","Peito"), 4, "Carga moderada"));

t.adicionarItem(new ItemTreino(new Exercicio("Remada Curvada","Costas"), 3, "Foco em técnica"));

**Treino** compõe vários **ItemTreino**, cada um parametrizado por um **Exercicio**.

**5) Tratamento de exceções de domínio**

try {

Aluno a = new Aluno("Maria", "123.456.789-00", new PlanoMensal("Básico", 99.90));

} catch (ValidacaoException e) {

System.out.println("Falha na criação do aluno: " + e.getMessage());

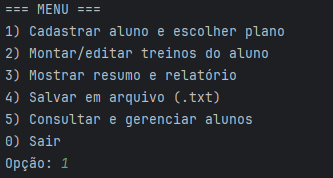
}

A **ValidacaoException** centraliza regras (ex.: CPF válido, valor de plano positivo).

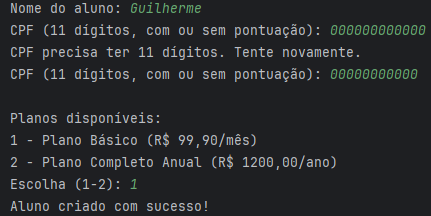
**Manual do Usuário (screenshots + instruções)**

Fluxo básico de uso

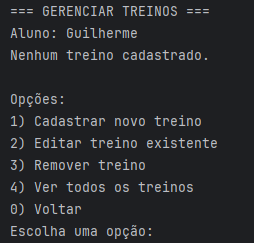
**1 – Cadastro de aluno:**



* 1. – Preenchimento de nome do aluno
  2. – Inserção de CPF (apenas números e 11 dígitos)
  3. – Escolha do plano (Básico/Completo)

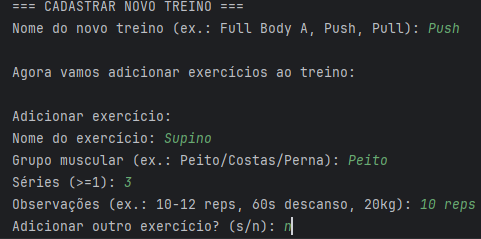


* + **2 – Montar/editar treinos do aluno:**

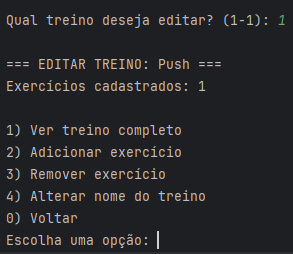


2.1 – Cadastro de novo treino

* + Para cada item: nome do exercício, grupo muscular, séries e observações (opcional).



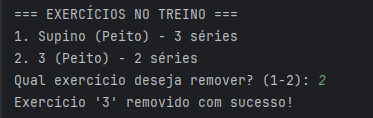
2.2 – Editar treino



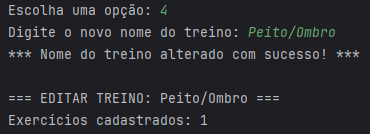
2.2.1 – Ver treino existente – informa as características do treino que já consta cadastrado

2.2.2 – Adicionar exercício – abre novamente a opção de adicionar um exercício ao treino

2.2.3 – Escolhe um exercício para ser removido do treino – com base no número do seu cadastro

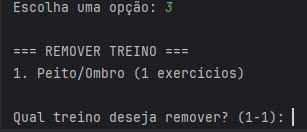


2.2.4 – Altera o nome dado ao treino que está sendo editado.



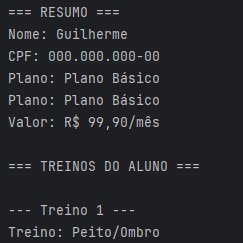
2.3 – Remover treino

Escolhe um treino cadastrado para sua remoção do sistema



**3 – Mostrar resumo e relatório**

Consulta os dados salvos no arquivo .txt do aluno que está sendo editado/consultado



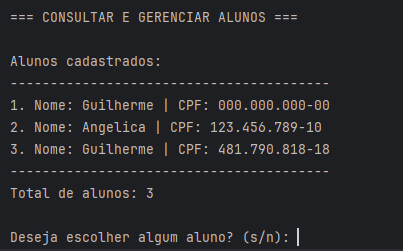
**4 – Salvar em arquivo**

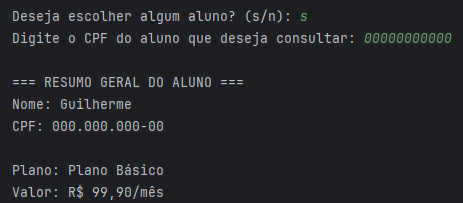
Salva as alterações feitas na ficha do aluno, gerando o .txt que será usado para consulta dos dados.



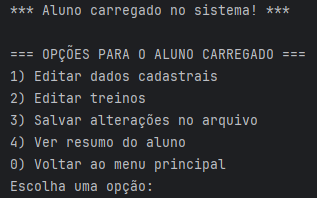
**5- Consultar aluno**

Te deixa escolher entre os alunos cadastrados, podendo carrega-ló através do txt para possíveis alterações de dados da ficha.

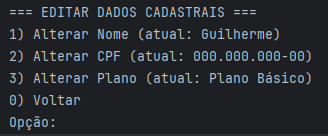




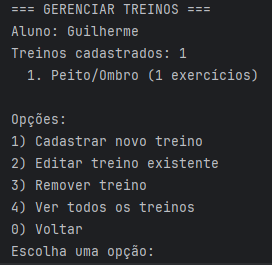
Ao carregar o aluno, te leva a um menu, com as seguintes opções:



5.1 – Editar dados cadastrais



5.2 – Gerenciar treinos, conforme mostrado na opção e edição generalizada anteriormente.



5.3 – Salva alterações do aluno que está sendo editado no seu respectivo arquivo.

5.4 – Novamente, gera um resumo dos dados do aluno. Feito para conferência das alterações que possam ter sido feitas.

**Link de repositório do git:**

<https://github.com/KaiqueJs/Projeto-final---POO.git>

**Conclusão**

O projeto consolida práticas de POO ao definir uma hierarquia clara (Pessoa → Aluno/Instrutor), uso de interfaces para variar comportamento (planos), composição para estruturar treinos, e exceções específicas para regras de negócio.  
O desenho por camadas de pacote e a documentação (UML + manual) favorecem manutenção e evolução — por exemplo, adicionar novos tipos de Plano ou permitir múltiplos treinos por Aluno sem quebrar código cliente.