RELATÓRIO TÉCNICO

Sistema de Gestão Acadêmica Orientado a Objetos (SOL)

Disciplina: Algoritmos e Estruturas de Dados
Projeto: Modelagem e Implementação em Java

2 de dezembro de 2025

Resumo

Este documento apresenta a documentação técnica do sistema desenvolvido para a gestão acadêmica simplificada, inspirado no sistema SOL. O projeto foi construído utilizando a linguagem Java, aplicando rigorosamente os quatro pilares da Programação Orientada a Objetos (POO): Abstração, Encapsulamento, Herança e Polimorfismo. O sistema opera via console e utiliza persistência de dados em memória através do padrão Singleton.

Sumário

1 Introdução

O objetivo deste projeto foi modelar e implementar um sistema capaz de gerenciar entidades acadêmicas fundamentais — Alunos, Professores e Turmas — sem a utilização de banco de dados relacional ou interface gráfica complexa. A ênfase recai sobre a estruturação correta das classes e seus relacionamentos.

2 Conceitos de POO Utilizados

A arquitetura do software baseia-se nos princípios fundamentais da orientação a objetos. Abaixo, detalhamos como cada conceito foi aplicado na prática.

2.1 Abstração

A abstração permitiu identificar as características essenciais de um indivíduo no contexto acadêmico, ignorando detalhes irrelevantes.

- **Implementação:** Foi criada a classe base `Pessoa`, que define o modelo genérico contendo *nome*, *cpf* e *email*. Esta classe não pode ser instanciada diretamente (`abstract`), servindo apenas como molde.

2.2 Herança

Utilizamos herança para promover a reutilização de código e estabelecer uma hierarquia lógica.

- **Implementação:** As classes `Aluno`, `Professor` e `Administrativo` estendem a classe `Pessoa`.
- **Benefício:** Evita-se a repetição dos atributos comuns (nome, cpf) em cada classe filha. O Java utiliza a palavra-chave `extends` para este fim.

```
1 public class Aluno extends Pessoa {
2     // Herda nome e cpf, e adiciona matricula
3     private String matricula;
4     ...
5 }
```

Listing 1: Exemplo de Herança no código

2.3 Polimorfismo

O polimorfismo permitiu tratar objetos de diferentes tipos de maneira uniforme, mas com comportamentos específicos.

- **Implementação:** O método abstrato `exibirDados()` foi definido na classe `Pessoa`. Cada subclasse implementa este método de forma distinta:
  - `Aluno`: Exibe Nome e Matrícula.
  - `Professor`: Exibe Nome e Titulação.
  - `Administrativo`: Exibe Nome e Cargo.

2.4 Encapsulamento

Para garantir a integridade dos dados, o acesso direto aos atributos foi restringido.

- **Implementação:** Todos os atributos são declarados como `private` ou `protected`. O acesso externo ocorre exclusivamente através de métodos públicos (Getters, Setters e Construtores), protegendo o estado interno do objeto contra alterações indevidas.

3 Funcionamento do Sistema

3.1 Arquitetura de Dados (Singleton)

Como o sistema não utiliza banco de dados externo, foi implementada a classe `BaseDeDadosMemoria` seguindo o padrão de projeto **Singleton**.

- Este padrão garante que exista apenas uma única instância da classe de banco de dados durante toda a execução do programa.
- Isso permite que os dados cadastrados pelo usuário *Administrativo* sejam acessados e visualizados em outras partes do sistema, mantendo a consistência em memória RAM.

3.2 Fluxo de Execução

O sistema é iniciado pela classe `SistemaAcademicoMain` e segue o seguinte fluxo:

1. **Autenticação:** O sistema solicita login. Apenas o perfil *Administrativo* (usuário: `user`, senha: `123`) possui permissão para realizar alterações (escrita).
2. **Menu Principal:** Após o login, é exibido um menu de opções no console:
    - **Cadastros:** Permite criar novos Alunos e Professores.
    - **Gestão de Turmas:** Permite criar turmas vinculando uma Disciplina e um ano letivo.
    - **Matrícula:** Associa um Aluno existente a uma Turma existente (Agregação).
    - **Relatórios:** Lista todos os dados cadastrados utilizando o método polimórfico `exibirDados()`.
3. **Relacionamentos:**
    - Um **Curso** é composto por várias **Disciplinas** (Composição).
    - Uma **Turma** agrega vários **Alunos** (Agregação).
    - Um **Professor** é associado a uma **Turma** como responsável.

4 Conclusão

O desenvolvimento deste sistema permitiu a aplicação prática dos conceitos teóricos de Orientação a Objetos. A separação de responsabilidades entre as classes de modelo (Entidades) e as classes de controle (Main e Banco de Dados) resultou em um código organizado, modular e de fácil manutenção, atendendo a todos os requisitos propostos na atividade.