# Programação Avançada - AULA 09

Matheus Moresco
Engenharia de Software - 5º Período
2025/01

# Aula Prática

- Projeto tem como objetivo criar um sistema acadêmico para gerenciar **Alunos, Professores e Curso** utilizando **POO e Estruturas de Dados em Java**.
- Vamos seguir um passo a passo para a implementação.

- Exercícios para complementação do projeto.

# Organização do projeto

- Criar a Estrutura do Projeto, organize seu projeto em pacotes para manter o código limpo e organizado.
- src/
  - model/ → Contém as classes principais (Pessoa, Aluno, Professor, Curso).
  - service/ → Contém a classe GestorAcademico, que gerencia os objetos.
  - App.java → Ponto de entrada do programa.

# Criar a Classe Abstrata Pessoa

- Objetivo: Servir como modelo para as classes Aluno e Professor, evitando código duplicado.
- Passos:
  1. Criar os atributos comuns (nome, idade, cpf).
  2. Criar um construtor para inicializar os atributos.
  3. Criar um método abstrato exibirDetalhes(), obrigatório nas classes filhas.

# Criar a Classe Aluno

- Objetivo: Representar um aluno com seus cursos.
- Passos:
  1. Extender a classe Pessoa.
  2. Criar o atributo RA.
  3. Criar uma lista de cursos para armazenar os cursos do aluno.
  4. Criar um método para matricular o aluno em um curso.
  5. Implementar exibirDetalhes() para exibir os cursos do aluno

# Criar a Classe Professor

- Objetivo: Representar um professor com especialidade e salário.
- Passos:
  1. Extender a classe Pessoa.
  2. Criar os atributos especialidade e salario.
  3. Implementar exibirDetalhes() para mostrar as informações do professor.

# Criar a Classe Curso

- Objetivo: Representar um curso.
- Passos:
    1. Criar os atributos nome e cargaHoraria.
    2. Criar um construtor para inicializar os atributos.
    3. Criar métodos getter para acessar os dados.

# Criar a Classe GestorAcademico

- Objetivo: Gerenciar alunos, professores e cursos.
- Passos:
  1. Criar listas para armazenar alunos e professores.
  2. Criar um mapa para armazenar cursos (Codigo → curso).
  3. Criar métodos para adicionar e buscar dados.
  4. Criar métodos para ordenar listas usando Streams.

# Criar a Classe Main (Executável)

- Objetivo: Criar objetos e testar o sistema.
- Passos:
  1. Criar um objeto GestorAcademico.
  2. Criar cursos, professores e alunos e adicioná-los ao gestor.
  3. Matricular alunos em cursos.
  4. Listar alunos e professores ordenados por nome.

# Exercício 1: Implementar a Remoção de Cursos para um Aluno

- Objetivo: Adicionar funcionalidade para um aluno remover um curso ao qual está matriculado.
- Tarefa:
  - Adicione um método desmatricularCurso(String nomeCurso) na classe Aluno.
  - Esse método deve remover o curso da lista de cursos matriculados.
  - No Main, teste a funcionalidade matriculando um aluno em dois cursos e depois removendo um deles.

# Exercício 2: Relatório de Professores e Seus Cursos

- Objetivo: Criar uma funcionalidade que relacione professores e os cursos que eles lecionam.
- Tarefa:
  - Adicione uma lista de cursos na classe Professor para armazenar os cursos que ele leciona.
  - Crie um método atribuirCurso(Curso curso) para associar um professor a um curso.
  - Adicione um método exibirCursos() que liste os cursos que ele ministra.
  - No Main, associe um professor a pelo menos dois cursos e exiba as informações.

# Exercício 3: Contar e Filtrar Alunos Matriculados em um Curso Específico

- Objetivo: Usar Streams para contar e listar alunos matriculados em um determinado curso.
- Tarefa:
  - No GestorAcademico, adicione um método listarAlunosPorCurso(String nomeCurso) que:
    - Filtra os alunos que estão matriculados no curso informado.
    - Retorna a lista desses alunos.
  - Crie um método contarAlunosPorCurso(String nomeCurso), que retorna o número total de alunos matriculados no curso.
  - No Main, teste listando alunos de um curso específico e exibindo a contagem.

# Exercício 4: Criar a Classe Disciplina e Relacioná-la com Aluno e Curso

1. Criar a classe Disciplina com os seguintes atributos:
   - nome (String) → Nome da disciplina
   - cargaHoraria (int) → Carga horária da disciplina em horas
   - periodo (int) → Período recomendado da disciplina
   - curso (Curso) → Curso ao qual a disciplina pertence
2. Modificar a classe Aluno para permitir a matrícula em disciplinas:
   - Adicionar um atributo disciplinasMatriculadas, que será uma lista de disciplinas.
   - Criar um método matricularDisciplina(Disciplina disciplina), que adiciona a disciplina à lista.
   - Criar um método exibirDisciplinasMatriculadas(), que lista todas as disciplinas do aluno.

# Exercício 4: Criar a Classe Disciplina e Relacioná-la com Aluno e Curso

3. Criar a relação entre Curso e Disciplina:
   - Adicionar um atributo disciplinas na classe Curso, que será uma lista de disciplinas.
   - Criar um método adicionarDisciplina(Disciplina disciplina), que vincula a disciplina ao curso.
   - Criar um método exibirDisciplinas(), que exibe todas as disciplinas do curso.
4. Testar no Main:
   - Criar um curso.
   - Criar algumas disciplinas vinculadas ao curso.
   - Criar um aluno e matriculá-lo em algumas disciplinas.
   - Exibir as disciplinas matriculadas do aluno.

# Envio do trabalho

- Prazo de envio: 06/04/2025
- Grupos de até 5 Pessoas.
- Envio via google form: https://forms.gle/AaVMaVkS33i3UutTA