Projeto de Banco de Dados: Sistema de Matrícula Escolar

Gerenciamento de Alunos via CLI (Java + MySQL)

Integrantes:
Pedro Paulo, Mariana Dias, Rafael Souza, Winiston Alle, Yuri Natanael

Disciplina / Matéria
Programação orientada a objetos (POO) /Samuel Novais Moura Junior

Data da Apresentação
31 de outubro de 2025

Escopo e Objetivo do Projeto

Objetivo

Desenvolver uma aplicação de console (CLI) em Java para realizar o gerenciamento completo (CRUD) de alunos em um banco de dados MySQL, demonstrando persistência de dados e arquitetura em camadas.

Projeto

Sistema de Matrícula Escolar

Tecnologias Utilizadas

- **Linguagem:** Java (compilado e executado via terminal)
- **Banco de Dados:** MySQL 8.0+
- **Conector:** JDBC (MySQL Connector/J 9.4.0)
- **Controle de Versão:** Git e GitHub

Restrições da Atividade

O projeto atende ao requisito de não utilizar tecnologias web (HTML, JSR, JSF), focando 100% na lógica de negócios e persistência de dados no console.

Arquitetura em Camadas: Visão e Controle

O projeto segue a arquitetura em camadas, separando responsabilidades em pacotes distintos que se comunicam de forma organizada. Apresentamos as duas primeiras camadas:

1

VISÃO

(Apresentação)

Interface com o usuário através do terminal. Exibe o menu, entradas do usuário e coordena as opções solicitadas.

Main.java

Ponto de entrada da aplicação. Gerencia a interação com o usuário através do terminal e coordena chamadas para a camada de controle.

Responsável por: Exibir menus, ler entradas, validar dados de entrada e coordenar fluxo de execução.

2

CONTROLE

(Lógica de Negócio)

Implementa toda a lógica CRUD. Administra as conexões e coordena com o banco de dados.

AlunoServico.java

Implementa a lógica de negócios. Processa dados, valida regras de negócio e coordena operações com o banco de dados.

Responsável por: Operações CRUD, validações de negócio, persistência de dados e gerenciamento de transações.

Arquitetura em Camadas: Modelo e Banco de Dados

3

MODELO

(Dados)

Define a estrutura de um aluno com atributos (id, nome, cpf, série, turno, telefone). Serve como modelo para objetos da classe Aluno, representando a entidade de dados no domínio da aplicação.

Aluno.java
Classe que representa a entidade Aluno. Armazena dados através de atributos privados acessíveis por métodos getters/setters, seguindo o padrão de encapsulamento Java.

4

BANCO DE DADOS

(Persistência)

Armazena permanentemente os dados dos alunos na tabela 'aluno'. Acessível através de comandos SQL (INSERT, SELECT, UPDATE, DELETE) para operações de persistência.

MySQL 8.0+
Banco de dados relacional que persiste dados através de operações SQL. Gerenciado pela classe Conexao que estabelece e mantém a conexão JDBC com segurança.

Próximo: Fluxo Completo de Dados e Benefícios da Arquitetura em Camadas

Fluxo de Dados e Benefícios da Arquitetura

Fluxo Completo de Dados: Exemplo Prático (Cadastrar Aluno)

1. **Usuário** escreve dados no terminal (CPF, nome, telefone)
2. **Main.java** (Visão) lê entrada do usuário via Scanner e coleta os dados
3. **AlunoServico.cadastrarAluno()** recebe o objeto Aluno e processa os dados
4. **Conexao.getConexao()** estabelece a conexão com MySQL
5. **SQL INSERT** é executado para salvar o aluno no banco de dados MySQL
6. **Mensagem de sucesso** é exibida no terminal para o usuário

Benefícios da Arquitetura em Camadas

- ✓ Separação de responsabilidades: cada camada tem função específica
- ✓ Facilita manutenção: mudanças isoladas em uma camada
- ✓ Reutilização de código: componentes reutilizáveis
- ✓ Testabilidade: cada camada testada independentemente

Pacote Modelo: Classe Aluno - Atributos (Parte 1)

A classe Aluno é o modelo que cria objetos que representam um aluno. Ela armazena dados através de atributos privados acessíveis por métodos getters/setters. Apresentamos os primeiros quatro atributos fundamentais:

int id

Identificador único, gerado automaticamente pelo banco (AUTO_INCREMENT). Chave Primária (PK).

Exemplo
id = 1, 2, 3, ...

String nome

Nome completo do aluno. Campo obrigatório.

Exemplo
"Pedro Amaral"

String cpf

CPF único do aluno (Chave Única - UNIQUE). Usado para buscas, edições e remoções.

Exemplo
"12345678901"

int idade

Idade do aluno em anos. Usado para controle administrativo.

Exemplo
idade = 10

Próximo Slide

Continuaremos com os três atributos restantes: **série** , **turno** e **telefone** , completando a estrutura completa da classe Aluno.

Pacote Modelo: Classe Aluno - Atributos

Continuação dos atributos fundamentais da classe Aluno. Apresentamos os três últimos atributos que completam a estrutura de dados:

String **série**

Série escolar do aluno. Define o nível de ensino.

Exemplo
"Fundamental 1", "Médio 2º Ano"

String **turno**

Turno de estudo do aluno. Define o horário das aulas.

Exemplo
"Matutino", "Vespertino"

String **telefone**

Número de telefone para contato da escola com a família do aluno.

Exemplo
“6199999000"

Resumo: Estes sete atributos (id, nome, cpf, idade, série, turno, telefone) formam a estrutura completa de dados da classe Aluno, sendo acessíveis através de métodos getters e setters seguindo o padrão Java.

Pacote Modelo: Classe Aluno - Getters e Setters

Os métodos getters e setters permitem acessar e modificar os atributos da classe Aluno. Cada atributo possui um par de métodos segundo o padrão de nomenclatura Java.

Getters (Leitura)

getId()
Retorna o ID do aluno

getNome()
Retorna o nome do aluno

getCpf()
Retorna o CPF do aluno

NÑPHŇVŇÑÄÅ
Retorna a idade do aluno

getSérie()
Retorna a série do aluno

getTurno()
Retorna o turno do aluno

getTelefone()
Retorna o telefone do aluno

Setters (Escrita)

setId(int)
Define o ID do aluno

setNome(String)
Define o nome do aluno

setCpf(String)
Define o CPF do aluno

setIdade(int)
Define a idade do aluno

setSérie(String)
Define a série do aluno

setTurno(String)
Define o turno do aluno

setTelefone(String)
Define o telefone do aluno

Padrão Getter/Setter: Cada atributo segue o padrão de nomenclatura Java com prefixos "get" e "set" seguidos do nome do atributo com primeira letra maiúscula.

Pacote Modelo: Classe Aluno - Métodos Especiais e Exemplo

Método toString()

Retorna uma representação legível de todos os dados do aluno em formato de string. Útil para exibir informações na tela.

```
publicStringtoString() {
return"ID: " + id + ", Nome: " + nome + ...
 }
```

Padrão de Nomenclatura Java

Getters: Cada método getter começa com o prefixo "get" seguido do nome do atributo com primeira letra maiúscula. Exemplo: getNome(), getCpf()

Setters: Cada método setter começa com o prefixo "set" seguido do nome do atributo com primeira letra maiúscula e recebe um parâmetro. Exemplo: setNome(String), setCpf(String)

Este padrão é essencial para o encapsulamento e facilita a leitura e manutenção do código Java.

Exemplo de Uso Prático

Demonstração completa de como criar um objeto Aluno, atribuir valores e exibir as informações:

```
Aluno aluno = newAluno();
 aluno.setNome("Pedro Amaral");
 aluno.setCpf("12345678901");
 aluno.setIdade(10);
 aluno.setSérie("Fundamental 1");
 aluno.setTurno("Matutino");
 aluno.setTelefone("5 199999000");
// Exibindo as informações do aluno
System.out.println(aluno.toString());
```

Pacote Controle: Classes Conexão e AlunoServico

O pacote controle implementa a lógica de negócios da aplicação. Duas classes trabalham em conjunto: uma gerencia a conexão com o banco de dados, outra implementa todas as operações CRUD.

Conexao.java

Classe utilitária responsável por estabelecer e gerenciar a conexão com o banco de dados MySQL.

Atributos (Constantes)

URL
Endereço do banco de dados

USER
Usuário do MySQL (root)

PASS
Senha do banco de dados

Método Principal

getConexao()
Retorna uma conexão ativa com o MySQL usando JDBC

AlunoServico.java

O "controller" da aplicação. Implementa toda a lógica CRUD e coordena operações entre a visão e o banco de dados.

Métodos CRUD

cadastrarAluno(Aluno)
INSERT - Salva um novo aluno no banco

listarAlunos()
SELECT * - Retorna lista com todos os alunos

buscarAlunoPorCpf(String)
SELECT WHERE - Busca um aluno específico

editarAluno(Aluno)
UPDATE - Modifica dados existentes

removerAluno(String)
DELETE - Remove aluno do banco

Pacote Visão: Classe Main - Conceitos Fundamentais

O que é a Classe Main?

É o **ponto de entrada** do programa. Quando você executa a aplicação, a Java chama pelo método **main()** nesta classe para iniciar a execução.

Responsável por gerenciar toda a **interação com o usuário** através do terminal, exibindo menus e coordenando as operações solicitadas.

Responsabilidades

- Exibir o menu ao usuário
- Ler entradas do usuário
- Chamar métodos da AlunoServico
- Manter o programa rodando
- Validar opções selecionadas
- Exibir mensagens de feedback

Método main() e Scanner

O método **main()** cria objetos importantes:

- **Scanner:** Lê dados do usuário
- **AlunoServico:** Métodos CRUD

```
public static void main(String[] args) { Scanner scanner = new Scanner(System.in); AlunoServico servico = new AlunoServico(); // ... resto do código }
```

Inicialização

A classe Main inicializa os componentes necessários para a aplicação funcionar:

- Cria instância do Scanner
- Cria instância do AlunoServico
- Inicia o loop principal
- Gerencia o ciclo de vida

Pacote Visão: Classe Main - Menu e Fluxo de Operações

Menu Interativo (do-while)

O programa usa um loop **do-while** para manter o menu rodando até o usuário escolher "Sair". Este padrão garante que o menu seja exibido pelo menos uma vez e continue se repetindo enquanto o usuário não encerrar a aplicação.

```
do { exibirMenu(); int opcao = scanner.nextInt(); processarOpcao(opcao); } while (continuar);
```

A variável **continuar** controla o fluxo do loop. Quando o usuário escolhe sair, ela é definida como **false**, encerrando a execução do programa.

Fluxo de Chamadas - Operações Disponíveis

O menu oferece seis operações principais que o usuário pode executar. Cada opção chama um método específico do **AlunoServico**:

1. **Criar Aluno:** servico.cadastrarAluno()
2. **Listar Alunos:** servico.listarAlunos()
4. **Editar Aluno:** servico.editarAluno()
5. **Remover Aluno:** servico.removerAluno()
3. **Buscar por CPF:** servico.buscarPorCpf()
6. **Sair:** Encerrar programa

História de Usuário: Secretário Acadêmico

Como um(a)

Secretário(a) Acadêmico(a)

Responsável pela **gestão de registros de alunos** e coordenação de informações educacionais na instituição.

Eu quero

Acessar um Sistema

Que me permita **Cadastrar, Buscar, Editar e Remover** alunos de forma **rápida e segura** através de uma interface de linha de comando.

Para que

Manter Dados Atualizados

Manter o **banco de dados da escola** sempre atualizado com **informações precisas e íntegras** dos alunos.

Próximo Slide

Veremos o contexto completo e as responsabilidades do secretário acadêmico no gerenciamento dos dados escolares.

História de Usuário: Contexto e Responsabilidades

Contexto do Sistema

O secretário acadêmico é responsável por gerenciar os registros de alunos, incluindo **matrículas, transferências, alterações de dados cadastrais e desligamentos**. O sistema deve permitir **conexões rápidas e confiáveis** para garantir a integridade dos dados em todas as operações.

Gerenciamento de Dados

- Gerenciar registros de alunos (matrículas, transferências, desligamentos)
- Manter dados cadastrais sempre atualizados e precisos
- Realizar buscas rápidas de informações de alunos
- Garantir a integridade e segurança dos dados

Relatórios e Coordenação

- Gerar relatórios e consultas conforme necessário
- Coordenar com outras áreas da instituição
- Manter registros atualizados em tempo real
- Validar integridade de informações cadastrais

Requisitos Técnicos do Sistema

O sistema deve fornecer uma **interface de linha de comando (CLI)** que permita operações **CRUD completas** (Criar, Ler, Atualizar, Deletar) com **validação de dados**, **persistência em banco de dados MySQL** e **feedback imediato ao usuário** sobre o sucesso ou falha de cada operação.

CRUD 1: CREATE - Entrada de Dados e Modelo

A operação **CREATE** permite adicionar um novo aluno ao banco de dados. O processo começa com a **entrada de dados do usuário** e a **criação do objeto Aluno**.

1. Entrada de Dados do Usuário

(Visão - Main.java)

O usuário escreve a opção "1" no menu. O programa solicita os dados do novo aluno através do terminal usando **Scanner**.

```
System.out.println("=== Cadastrar Novo Aluno ==="); System.out.print("Nome: ");
String nome = scanner.nextLine(); System.out.print("CPF: "); String cpf =
scanner.nextLine(); System.out.print("Idade: "); int idade = scanner.nextInt();
scanner.nextLine(); System.out.print("Série: "); String serie = scanner.nextLine();
System.out.print("Turno: "); String turno = scanner.nextLine();
System.out.print("Telefone: "); String telefone = scanner.nextLine();
```

Entrada: Dados digitados pelo usuário no terminal

2. Criação do Objeto Aluno

(Modelo - Aluno.java)

Após coletar todos os dados, o programa cria um novo objeto **Aluno** e atribui os valores coletados usando os métodos **setters**.

```
Aluno novoAluno = new Aluno(); novoAluno.setNome(nome);
novoAluno.setCpf(cpf); novoAluno.setIdade(idade);
novoAluno.setSérie(serie); novoAluno.setTurno(turno);
novoAluno.setTelefone(telefone);
```

Saída: Objeto Aluno completo e pronto para persistência

Próximo Slide

Veremos o processamento dos dados e a persistência no banco de dados MySQL.

CRUD 1: CREATE - Processamento e Persistência

3 Processamento (AlunoServico)

O **AlunoServico** recebe o objeto Aluno criado pela visão e processa os dados antes de persistir no banco.

```
public void cadastrarAluno(Aluno aluno) { try { // Validações if (aluno.getNome().isEmpty()) { System.out.println("Erro: Nome vazio!"); return; } // Inserir no banco inserirAluno(aluno); } catch (Exception e) { System.out.println("Erro: " + e.getMessage()); } }
```

O método **cadastrarAluno()** valida os dados e chama o método **inserirAluno()** para persistir no banco de dados.

4 Persistência (Banco de Dados)

O método **inserirAluno()** executa um comando **SQL INSERT** para salvar permanentemente o aluno no banco de dados MySQL.

```
private void inserirAluno(Aluno aluno) { String sql = "INSERT INTO aluno " + "(nome, cpf, idade, serie, turno, telefone) " + "VALUES (?, ?, ?, ?, ?, ?)"; PreparedStatement stmt = conexao.prepareStatement(sql); stmt.setString(1, aluno.getNome()); stmt.setString(2, aluno.getCpf()); // ... demais parâmetros stmt.executeUpdate(); }
```

o banco retorna o ID gerado automaticamente (AUTO_INCREMENT) para o novo aluno.

5 Resultado e Confirmação

Sucesso:

✓Aluno cadastrado com sucesso!
ID gerado: 1

Banco de Dados:

Novo registro inserido na tabela 'aluno'
Dados persistidos permanentemente

CRUD 2 e 3: READ - Buscar e Listar Alunos

A operação **READ** (Leitura) consulta dados existentes no banco de dados. O sistema oferece duas formas de leitura: **listar todos os alunos** ou **buscar um aluno específico por CPF**.

Método listarAlunos()

Retorna uma lista com todos os alunos cadastrados no banco de dados. Executa um SELECT sem filtros.

```
SELECT * FROM aluno;
```

Resultado: Lista completa de alunos com todos os seus atributos.

Método buscarPorCpf(String cpf)

Busca um aluno específico usando o CPF como critério. Retorna um único aluno ou null se não encontrado.

```
SELECT * FROM aluno WHERE cpf = ?;
```

Resultado: Dados completos de um aluno específico.

Sequência de Execução (READ)

1. Usuário escolhe opção "2" (Listar) ou “3" (Buscar por CPF) no menu
2. Main.java coleta dados (se necessário) e chama AlunoServico
3. AlunoServico executa consulta SQL no banco de dados MySQL
4. Banco retorna resultado (lista ou objeto Aluno)
5. Main.java exibe os dados formatados no terminal para o usuário

Exemplo Prático
Entrada do Usuário:
Opção: 3 (Buscar por CPF)
CPF: "12345678901"

Resultado Esperado:
ID: 1
Nome: Pedro Amaral
CPF: 12345678901
Idade: 10

CRUD 4: UPDATE - Editar Aluno

A operação **UPDATE** permite modificar dados de um aluno existente. O processo envolve **buscar o aluno**, **coletar novos dados** e **persistir as alterações** no banco de dados.

1 Buscar Aluno Existente

O usuário escolhe a opção “4" no menu. O programa solicita o **CPF do aluno** a ser editado e busca seus dados no banco de dados.

```
System.out.print("Digite o CPF do aluno a editar: "); String cpf = scanner.nextLine(); Aluno aluno = buscarAlunoPorCpf(cpf); if (aluno == null) { System.out.println("Aluno não encontrado!"); return; } System.out.println("Aluno encontrado: " + aluno.getNome());
```

2 Coleta de Novos Dados

O programa exibe os dados atuais e solicita os **novos valores** para os atributos que o usuário deseja modificar.

```
System.out.print("Novo nome (atual: " + aluno.getNome() + "): "); aluno.setNome(scanner.nextLine()); System.out.print("Novo turno (atual: " + aluno.getTurno() + "): "); aluno.setTurno(scanner.nextLine()); System.out.print("Novo telefone (atual: " + aluno.getTelefone() + "): "); aluno.setTelefone(scanner.nextLine());
```

3 Atualização no Banco de Dados

O **AlunoServico** executa um comando **SQL UPDATE** para persistir as alterações no banco de dados MySQL.

```
String sql = "UPDATE aluno SET nome=?, turno=?, telefone=? WHERE cpf=?"; PreparedStatement stmt = conexao.prepareStatement(sql); stmt.setString(1, aluno.getNome()); stmt.setString(2, aluno.getTurno()); stmt.setString(3, aluno.getTelefone()); stmt.setString(4, aluno.getCpf()); stmt.executeUpdate();
```

CRUD 5: DELETE - Remover Aluno

A operação **DELETE** remove permanentemente um aluno do banco de dados. Esta é uma operação **direta e irreversível** que é executada imediatamente após a identificação do registro .

1 Busca do Aluno

O usuário escolhe a opção “5" (Remover Aluno) no menu. O programa solicita o **CPF** do aluno a ser removido e busca seus dados no banco de dados.

```
System.out.print("Digite o CPF do aluno a remover: "); String cpf = scanner.nextLine(); Aluno aluno = servico.buscarPorCpf(cpf); if (aluno == null) {
System.out.println("Aluno não encontrado!"); return; }
```

2 Execução Imediata do DELETE

Após a entrada do CPF, o sistema executa imediatamente o comando SQL DELETE para remover permanentemente o aluno do banco de dados:

```
String sql = "DELETE FROM aluno WHERE cpf = ?";
try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
    stmt.setString(1, cpf);
    int resultado = stmt.executeUpdate();
    // ... (continua no passo 3)
} catch (SQLException e) {
    e.printStackTrace();
```

3 Confirmação de Sucesso/Falha

O sistema verifica o resultado da execução do comando SQL e exibe a mensagem correspondente.
Após confirmação, o sistema executa o comando SQL DELETE para remover permanentemente o aluno do banco de dados.

Conclusão: Integração e Perspectivas Futuras

Síntese do Sistema Integrado

O **Sistema de Matrícula Escolar** demonstra uma arquitetura em camadas bem estruturada, onde a **Visão (Main)** interage com o usuário, a **Controle (AlunoServico)** processa as operações, o **Modelo (Aluno)** representa os dados e o **Banco de Dados (MySQL)** persiste as informações. Todas as operações **CRUD** (Criar, Ler, Atualizar, Deletar) funcionam de forma integrada e segura, garantindo a integridade dos dados escolares.

Conceitos Fundamentais Demonstrados

Arquitetura em Camadas

Separação clara de responsabilidades entre visão, controle, modelo e persistência

Operações CRUD

Implementação completa de Create, Read, Update e Delete com validações

Padrões Java

Getters/Setters, encapsulamento, POO e boas práticas de desenvolvimento

Banco de Dados Relacional

Persistência com MySQL, SQL, JDBC e gerenciamento de conexões

Interface de Usuário CLI

Menu interativo com Scanner, validação de entrada e feedback ao usuário

Segurança de Dados

Validação de CPF único, confirmação de exclusão e integridade referencial

Perspectivas de Expansão Futura

Interface Gráfica (GUI)

Migração para JavaFX ou Swing para melhor experiência do usuário

Autenticação e Autorização

Sistema de login com diferentes níveis de acesso (admin, secretário, professor)