

# Explicação Detalhada do Sistema de Biblioteca UNIPE

## Visão Geral do Projeto

Este é um sistema completo de gerenciamento de biblioteca desenvolvido em Java com interface via terminal, utilizando MySQL como banco de dados. O sistema foi projetado para atender às necessidades de uma biblioteca universitária, com funcionalidades específicas para bibliotecários e alunos.

## Arquitetura do Sistema

### 1. Estrutura de Arquivos

```
BibliotecaUNIPE/
├── lib/
│   └── mysql-connector-j-9.3.0.jar
├── src/
│   ├── Biblioteca/
│   │   ├── conexao/
│   │   │   ├── ConexaoMySQL.class
│   │   │   └── ConexaoMySQL.java
│   │   ├── BDbiblioteca.sql
│   │   ├── Aluno.java
│   │   ├── AppBiblioteca.java
│   │   ├── Emprestimo.java
│   │   ├── Livro.java
│   │   └── Metodos.java
│   └── README.md
```

### 2. Componentes Principais

#### a) Banco de Dados (BDbiblioteca.sql)

O banco de dados MySQL contém 7 tabelas principais:

1. **Cliente:** Armazena informações dos alunos (RGM, nome, endereço, email)
2. **Livro:** Contém dados dos livros (ID, título, autor, ano, gênero, status)
3. **Bibliotecario:** Registra os bibliotecários do sistema
4. **Emprestimo:** Gerencia os empréstimos (status, datas)
5. **ClienteEmprestimo:** Tabela de junção entre Cliente e Empréstimo
6. **EmprestimoLivro:** Tabela de junção entre Empréstimo e Livro
7. **SupervisaoCliente:** Relaciona bibliotecários com clientes

## b) Classes Java

1. **Aluno.java**: Modela um aluno com RGM, nome, endereço e email
2. **Livro.java**: Representa um livro com ID, título, autor, ano, gênero e status
3. **Emprestimo.java**: Modela um empréstimo com ID, RGM do aluno, ID do livro, datas e status
4. **ConexaoMySQL.java**: Gerencia a conexão com o banco de dados
5. **Metodos.java**: Contém toda a lógica de negócios do sistema
6. **AppBiblioteca.java**: Classe principal com a interface do usuário

## Configuração e Execução Passo a Passo

### 1. Pré-requisitos

- Java JDK 8 ou superior instalado
- MySQL Server instalado e em execução
- MySQL Workbench (opcional para gerenciamento visual)

### 2. Configuração do Banco de Dados

#### 1. Execute o script `BDbiblioteca.sql` no MySQL:

```
```bash
```

```
mysql -u root -p < src/Biblioteca/BDbiblioteca.sql
```

```
``` (Substitua "root" pelo seu usuário MySQL)
```

#### 2. Verifique se o banco foi criado corretamente com as tabelas e dados iniciais

### 3. Configuração da Conexão

Edite o arquivo `ConexaoMySQL.java` para refletir suas credenciais:

```
```java
```

```
private static final String URL = "jdbc:mysql://localhost:3306/BDbiblioteca";  
private static final String USER = "root"; // seu usuário MySQL  
private static final String PASSWORD = ""; // sua senha MySQL
```

```
```
```

## 4. Compilação e Execução

1. Navegue até a pasta raiz do projeto (onde está a pasta src)

2. Compile o projeto:

```
```bash
```

```
javac -cp ./lib/mysql-connector-j-9.3.0.jar src/Biblioteca/AppBiblioteca.java
```

```
```
```

3. Execute o sistema:

```
```bash
```

```
java -cp ./lib/mysql-connector-j-9.3.0.jar Biblioteca.AppBiblioteca
```

```
```
```

## Explicação Detalhada do Código

### 1. ConexaoMySQL.java

Esta classe gerencia a conexão com o banco de dados:

```
```java
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConexaoMySQL {

    // URL de conexão com o banco de dados MySQL
    private static final String URL = "jdbc:mysql://localhost:3306/BDbiblioteca";

    // Usuário do banco de dados
    private static final String USER = "root";

    // Senha do banco de dados
    private static final String PASSWORD = "";

    // Método estático para obter uma conexão com o banco
    public static Connection getConexao() throws SQLException {
        try {
            // Carrega o driver JDBC do MySQL (necessário para versões mais antigas)
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Retorna uma conexão utilizando URL, usuário e senha definidos
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

\_\_\_\_\_

///

## 2. AppBiblioteca.java

Classe principal que implementa a interface do usuário:

```
```java
```

```
// Declaração da classe principal da aplicação
public class AppBiblioteca {

    // Scanner para entrada de dados via teclado (reutilizado em todo o programa)
    private static final Scanner scanner = new Scanner(System.in);

    // Método principal (ponto de entrada do programa)
    public static void main(String[] args) {
        int perfil; //Variável para armazenar a escolha do usuário

        // Loop principal do menu até o usuário escolher sair (opção 0)
        do {
            // Exibe o menu de perfis
            System.out.println("\n=== SISTEMA DE BIBLIOTECA ===");
            System.out.println("1. Bibliotecário");
            System.out.println("2. Aluno");
            System.out.println("0. Sair");
            System.out.print("Escolha: ");

            try {
                // Lê a entrada do usuário como string e converte para inteiro
                perfil = Integer.parseInt(scanner.nextLine());
            } catch (NumberFormatException e) {
                System.out.println("Entrada inválida. Por favor, digite um número inteiro.");
            }
        } while (perfil != 0);
    }
}
```

```

    } catch (NumberFormatException e) {
        // Se o usuário digitar algo inválido (ex: letra), assume -1 como escolha inválida
        perfil = -1;
    }

    // Executa ação de acordo com a escolha do usuário
    switch (perfil) {
        case 1 -> menuBibliotecario(); // Chama menu do bibliotecário
        case 2 -> menuAluno(); // Chama menu do aluno
        case 0 -> System.out.println("Saindo do sistema..."); // Encerra o sistema
        default -> System.out.println("Opção inválida. Tente novamente."); // Qualquer
        outra entrada
    }

    } while (perfil != 0); // Repete o menu até o usuário digitar 0
}

// Placeholder para o menu do bibliotecário
public static void menuBibliotecario() {
    System.out.println(">>> Menu do Bibliotecário (em desenvolvimento)");
    // Aqui você pode adicionar opções como: cadastrar livro, listar livros, etc.
}

// Placeholder para o menu do aluno
public static void menuAluno() {
    System.out.println(">>> Menu do Aluno (em desenvolvimento)");
    // Aqui você pode adicionar opções como: visualizar livros disponíveis, fazer
    empréstimo, etc.
}
}
}

```

### 3. Fluxo Principal

1. **Menu Inicial:** O usuário escolhe entre acessar como bibliotecário ou aluno

2. **Menu do Bibliotecário:**

- Gerenciar Alunos (CRUD completo)
- Gerenciar Livros (CRUD completo)
- Gerenciar Empréstimos (criar, finalizar, listar)
- Consultas avançadas (filtrar por gênero, autor, status, etc.)

3. **Menu do Aluno:**

- Fazer empréstimo
- Consultar histórico
- Ver livros disponíveis

## 4. Métodos.java - Principais Funcionalidades

### a) Gerenciamento de Alunos

```java

```
/**
 * Realiza o cadastro de um novo aluno no sistema.
 *
 * O método solicita o RGM (validado com 8 dígitos numéricos),
 * verifica se já está cadastrado, coleta nome, endereço e email,
 * e insere os dados na tabela Cliente no banco de dados.
 *
 * Requisitos:
 * - RGM deve ser único e conter exatamente 8 dígitos.
 * - Nome, endereço e e-mail são coletados diretamente do usuário.
 *
 * Exibe mensagens informativas em caso de sucesso ou erro.
 */
public static void cadastrarAluno() {
    try {
        // Declaração do RGM do aluno
        int rgm;

        // Validação de entrada do RGM
        while (true) {
            System.out.print("Digite o RGM do aluno (8 dígitos): ");
            String rgmInput = scanner.nextLine(); // Lê entrada como string

            // Verifica se o RGM tem 8 dígitos
            if (rgmInput.length() != 8) {
                System.out.println("RGM deve ter exatamente 8 dígitos!");
                continue; // Volta para o início do loop
            }

            try {
                // Tenta converter para inteiro
                rgm = Integer.parseInt(rgmInput);
                break; // Saída do loop se a conversão for bem-sucedida
            } catch (NumberFormatException e) {
                // Mostra erro caso contenha letras ou símbolos
                System.out.println("RGM deve conter apenas números!");
            }
        }

        // Verifica se o RGM já existe no banco
        if (alunoExiste(rgm)) {
            System.out.println("Erro: Este RGM já está cadastrado!");
            return; // Interrompe o método
        }

        // Coleta dos demais dados do aluno
    }
}
```

```

System.out.print("Digite o nome do aluno: ");
String nome = scanner.nextLine();

System.out.print("Digite o endereço do aluno: ");
String endereco = scanner.nextLine();

System.out.print("Digite o e-mail do aluno: ");
String email = scanner.nextLine();

// Inserção no banco de dados com uso de PreparedStatement
try (Connection conn = ConexaoMySQL.getConexao();
    PreparedStatement stmt = conn.prepareStatement(
        "INSERT INTO Cliente (RGM, Nome, Endereco, Email) VALUES (?, ?, ?, ?)")) {

    // Define os valores dos parâmetros na consulta SQL
    stmt.setInt(1, rgm);
    stmt.setString(2, nome);
    stmt.setString(3, endereco);
    stmt.setString(4, email);

    // Executa a inserção
    int linhasAfetadas = stmt.executeUpdate();

    // Verifica se a inserção foi bem-sucedida
    if (linhasAfetadas > 0) {
        System.out.println("\n✅ Aluno cadastrado com sucesso!");
    }
} catch (SQLException e) {
    // Tratamento de exceção de banco de dados
    System.err.println("❌ Erro no banco de dados: " + e.getMessage());
}
}

```

## b) Gerenciamento de Livros

```

```java

```

```

/**
 * Realiza o cadastro de um novo livro no banco de dados.
 *
 * O método coleta os dados básicos do livro (título, autor, ano, gênero),
 * realiza a validação (quando implementada), e insere o registro na tabela Livro.
 * O status do livro é definido como "Disponível" por padrão.
 *
 * Em caso de sucesso, exibe uma mensagem positiva.
 * Em caso de erro no banco, exibe a mensagem de erro correspondente.
 */
public static void cadastrarLivro() {
    try {
        // Declaração das variáveis que armazenarão os dados do livro
    }
}

```

```

String titulo, autor, genero;
int ano;

// Aqui você pode adicionar validações interativas:
// Exemplo de coleta com validação simples:
System.out.print("Digite o título do livro: ");
titulo = scanner.nextLine();

System.out.print("Digite o autor do livro: ");
autor = scanner.nextLine();

System.out.print("Digite o ano de publicação: ");
try {
    ano = Integer.parseInt(scanner.nextLine());
} catch (NumberFormatException e) {
    System.out.println("Ano inválido. Cadastro cancelado.");
    return;
}

System.out.print("Digite o gênero do livro: ");
genero = scanner.nextLine();

// Conexão com o banco de dados e inserção do novo livro
try (Connection conn = ConexaoMySQL.getConexao();
    PreparedStatement stmt = conn.prepareStatement(
        "INSERT INTO Livro (Titulo, Autor, Ano, Genero, Status) VALUES (?, ?, ?, ?,
'Disponível')")) {

    // Define os parâmetros da instrução SQL
    stmt.setString(1, titulo); // Título do livro
    stmt.setString(2, autor); // Autor do livro
    stmt.setInt(3, ano); // Ano de publicação
    stmt.setString(4, genero); // Gênero literário

    // Executa a operação de inserção no banco
    int linhasAfetadas = stmt.executeUpdate();

    // Verifica se a operação teve sucesso
    if (linhasAfetadas > 0) {
        System.out.println("✅ Livro cadastrado com sucesso!");
    }
} catch (SQLException e) {
    // Trata qualquer erro relacionado ao banco de dados
    System.err.println("❌ Erro no banco de dados: " + e.getMessage());
}
}

```

### c) Gerenciamento de Empréstimos

```

```java

```



```

/**
 * Registra um novo empréstimo de livro para um aluno específico.
 *
 * Este método realiza as seguintes operações dentro de uma transação:
 * 1. Cria um novo registro de empréstimo na tabela `Emprestimo`.
 * 2. Obtém o ID do empréstimo gerado automaticamente.
 * 3. Relaciona o empréstimo ao aluno na tabela `ClienteEmprestimo`.
 * 4. Relaciona o empréstimo ao livro na tabela `EmprestimoLivro`.
 * 5. Atualiza o status do livro para "Emprestado" na tabela `Livro`.
 *
 * Em caso de falha em qualquer uma das etapas, a transação é revertida.
 *
 * @param rgm RGM do aluno (identificador do cliente).
 * @param idLivro ID do livro a ser emprestado.
 * @return true se o empréstimo foi registrado com sucesso.
 * @throws SQLException caso ocorra erro na conexão ou execução SQL.
 */
public static boolean registrarEmprestimo(int rgm, int idLivro) throws SQLException {
    Connection conn = null; // Conexão com o banco de dados

    try {
        conn = ConexaoMySQL.getConexao(); // Abre conexão com o banco
        conn.setAutoCommit(false); // Inicia transação manualmente (desativa
autocommit)

        // 1. Cria novo empréstimo na tabela Emprestimo
        try (PreparedStatement stmtEmprestimo = conn.prepareStatement(
            "INSERT INTO Emprestimo (Status, DataRetirada, DataDevolucao) " +
            "VALUES ('Ativo', CURDATE(), DATE_ADD(CURDATE(), INTERVAL 7 DAY))",
            PreparedStatement.RETURN_GENERATED_KEYS)) { // Solicita retorno da chave
            gerada (ID do empréstimo)

            stmtEmprestimo.executeUpdate(); // Executa o INSERT

            // 2. Recupera o ID do empréstimo recém-criado
            int idEmprestimo;
            try (ResultSet rs = stmtEmprestimo.getGeneratedKeys()) {
                if (rs.next()) {
                    idEmprestimo = rs.getInt(1); // Pega o ID gerado
                } else {
                    throw new SQLException("Falha ao obter ID do empréstimo");
                }
            }

            // 3. Insere relação Cliente ↔ Empréstimo
            try (PreparedStatement stmtClienteEmprestimo = conn.prepareStatement(
                "INSERT INTO ClienteEmprestimo (RGM_Cliente, ID_Emprestimo) VALUES (?,
?))") {
                stmtClienteEmprestimo.setInt(1, rgm); // Define o RGM do aluno
                stmtClienteEmprestimo.setInt(2, idEmprestimo); // Associa com o empréstimo
                stmtClienteEmprestimo.executeUpdate();
            }
        }
    }
}

```

```

// 4. Insere relação Empréstimo ↔ Livro
try (PreparedStatement stmtEmprestimoLivro = conn.prepareStatement(
    "INSERT INTO EmprestimoLivro (ID_Livro, ID_Emprestimo) VALUES (?, ?)") {
    stmtEmprestimoLivro.setInt(1, idLivro);    // Define o ID do livro
    stmtEmprestimoLivro.setInt(2, idEmprestimo); // Associa com o empréstimo
    stmtEmprestimoLivro.executeUpdate();
}

// 5. Atualiza status do livro para "Emprestado"
try (PreparedStatement stmtAtualizarLivro = conn.prepareStatement(
    "UPDATE Livro SET Status = 'Emprestado' WHERE ID = ?") {
    stmtAtualizarLivro.setInt(1, idLivro);
    stmtAtualizarLivro.executeUpdate();
}

conn.commit(); // Confirma todas as alterações (transação bem-sucedida)
return true; // Retorna sucesso

} catch (SQLException e) {
    if (conn != null) {
        conn.rollback(); // Reverte todas as operações em caso de erro
    }
    throw e; // Relança a exceção para tratamento externo
} finally {
    if (conn != null) {
        conn.setAutoCommit(true); // Restaura comportamento padrão da conexão
        conn.close();           // Fecha conexão com o banco
    }
}
}
}

```

## 5. Classes de Modelo do Sistema de Biblioteca

### a) Aluno.java

```
```java
```

```

package Biblioteca;

/**
 * Representa um aluno cadastrado no sistema da biblioteca.
 * Contém informações básicas como RGM, nome, endereço e email.
 */
public class Aluno {
    // Atributos privados para encapsulamento
    private int rgm;    // Registro Geral do Aluno (identificador único)
    private String nome; // Nome completo do aluno
    private String endereco; // Endereço residencial do aluno
    private String email; // Email do aluno
}

```

```
/**
 * Construtor da classe Aluno.
 * Inicializa o objeto com os dados fornecidos.
 *
 * @param rgm    Identificador único do aluno (RGM).
 * @param nome    Nome completo do aluno.
 * @param endereco Endereço residencial do aluno.
 * @param email   Email do aluno.
 */
public Aluno(int rgm, String nome, String endereco, String email) {
    this.rgm = rgm;        // Inicializa o RGM do aluno
    this.nome = nome;      // Inicializa o nome do aluno
    this.endereco = endereco; // Inicializa o endereço do aluno
    this.email = email;    // Inicializa o email do aluno
}

// Métodos getters para acessar os atributos privados

/**
 * Retorna o RGM do aluno.
 * @return rgm do aluno
 */
public int getRgm() {
    return rgm;
}

/**
 * Retorna o nome do aluno.
 * @return nome do aluno
 */
public String getNome() {
    return nome;
}

/**
 * Retorna o endereço do aluno.
 * @return endereço do aluno
 */
public String getEndereco() {
    return endereco;
}

/**
 * Retorna o email do aluno.
 * @return email do aluno
 */
public String getEmail() {
    return email;
}

// Métodos setters para modificar atributos mutáveis
```

```

/**
 * Define um novo nome para o aluno.
 * @param nome novo nome
 */
public void setNome(String nome) {
    this.nome = nome;
}

/**
 * Define um novo endereço para o aluno.
 * @param endereco novo endereço
 */
public void setEndereco(String endereco) {
    this.endereco = endereco;
}

/**
 * Define um novo email para o aluno.
 * @param email novo email
 */
public void setEmail(String email) {
    this.email = email;
}
}

```

#### Explicação:

#### Atributos:

- `rgm` : Número de identificação único do aluno (8 dígitos) - **imutável**
- `nome` : Nome completo do aluno
- `endereco` : Endereço residencial
- `email` : Endereço eletrônico válido

#### Métodos:

- **Construtor:** Inicializa todos os atributos obrigatórios
- **Getters:** Permitem acesso aos valores dos atributos
- **Setters:** Apenas para campos mutáveis (nome, endereço, email). O RGM não tem setter pois é imutável.

#### b) Livro.java

```

```java

```

```

package Biblioteca;

/**
 * Representa um livro cadastrado na biblioteca.
 * Contém informações sobre o livro, como id, título, autor, ano, gênero e status.
 */

```

```

public class Livro {
    // Atributos privados para encapsulamento dos dados do livro
    private int id;      // Identificador único do livro
    private String titulo; // Título do livro
    private String autor; // Autor do livro
    private int ano;     // Ano de publicação do livro
    private String genero; // Gênero literário do livro
    private String status; // Status do livro (e.g., "Disponível", "Emprestado")

    /**
     * Construtor da classe Livro.
     * Inicializa o objeto com os dados fornecidos.
     *
     * @param id      Identificador único do livro
     * @param titulo  Título do livro
     * @param autor   Autor do livro
     * @param ano     Ano de publicação
     * @param genero  Gênero literário
     * @param status  Status atual do livro
     */
    public Livro(int id, String titulo, String autor, int ano, String genero, String status) {
        this.id = id;      // Inicializa o ID do livro
        this.titulo = titulo; // Inicializa o título do livro
        this.autor = autor; // Inicializa o autor do livro
        this.ano = ano;     // Inicializa o ano de publicação
        this.genero = genero; // Inicializa o gênero do livro
        this.status = status; // Inicializa o status do livro
    }

    // Métodos getters para acessar os atributos privados

    /**
     * Retorna o ID do livro.
     * @return id do livro
     */
    public int getId() {
        return id;
    }

    /**
     * Retorna o título do livro.
     * @return título do livro
     */
    public String getTitulo() {
        return titulo;
    }

    /**
     * Retorna o autor do livro.
     * @return autor do livro
     */
    public String getAutor() {

```

```

        return autor;
    }

    /**
     * Retorna o ano de publicação do livro.
     * @return ano do livro
     */
    public int getAno() {
        return ano;
    }

    /**
     * Retorna o gênero do livro.
     * @return gênero do livro
     */
    public String getGenero() {
        return genero;
    }

    /**
     * Retorna o status atual do livro.
     * @return status do livro
     */
    public String getStatus() {
        return status;
    }

    // Setter para o status, pois é o único campo que pode mudar após criação

    /**
     * Define um novo status para o livro.
     * @param status novo status (e.g., "Disponível", "Emprestado")
     */
    public void setStatus(String status) {
        this.status = status;
    }
}

```

### Explicação:

#### Atributos:

- `id`: Identificador único do livro - **imutável**
- `titulo`: Título completo do livro
- `autor`: Nome do autor
- `ano`: Ano de publicação
- `genero`: Categoria/gênero literário
- `status`: Pode ser "Disponível", "Emprestado" ou "Em manutenção"

#### Métodos:

- **Construtor:** Inicializa todos os atributos
- **Getters:** Permitem acesso a todos os atributos
- **Setter:** Apenas para `status`, pois é o único campo que pode mudar após a criação

### c) Emprestimo.java

```java

```
package Biblioteca;

import java.util.Date;

/**
 * Representa um empréstimo de livro realizado por um aluno na biblioteca.
 * Contém informações sobre o empréstimo, incluindo datas e status.
 */
public class Emprestimo {
    // Atributos privados que armazenam os dados do empréstimo

    private int id;          // Identificador único do empréstimo
    private int rgmAluno;    // RGM (identificador) do aluno que realizou o empréstimo
    private int idLivro;     // ID do livro emprestado
    private Date dataRetirada; // Data em que o livro foi retirado
    private Date dataDevolucao; // Data prevista para devolução do livro
    private String status;   // Status atual do empréstimo (ex: "Ativo", "Finalizado")

    /**
     * Construtor da classe Emprestimo.
     * Inicializa o empréstimo com os dados fornecidos.
     */
    @param id          Identificador único do empréstimo
    @param rgmAluno    RGM do aluno que fez o empréstimo
    @param idLivro     ID do livro emprestado
    @param dataRetirada Data de retirada do livro
    @param dataDevolucao Data prevista para devolução do livro
    @param status      Status atual do empréstimo
    /**
    public Emprestimo(int id, int rgmAluno, int idLivro, Date dataRetirada, Date
dataDevolucao, String status) {
        this.id = id;          // Inicializa o ID do empréstimo
        this.rgmAluno = rgmAluno; // Inicializa o RGM do aluno
        this.idLivro = idLivro; // Inicializa o ID do livro
        this.dataRetirada = dataRetirada; // Inicializa a data de retirada
        this.dataDevolucao = dataDevolucao; // Inicializa a data de devolução prevista
        this.status = status;   // Inicializa o status do empréstimo
    }

    // Getters para acessar os atributos privados
}
```

```
/**
 * Retorna o ID do empréstimo.
 * @return id do empréstimo
 */
public int getId() {
    return id;
}

/**
 * Retorna o RGM do aluno que realizou o empréstimo.
 * @return RGM do aluno
 */
public int getRgmAluno() {
    return rgmAluno;
}

/**
 * Retorna o ID do livro emprestado.
 * @return ID do livro
 */
public int getIdLivro() {
    return idLivro;
}

/**
 * Retorna a data de retirada do livro.
 * @return data de retirada
 */
public Date getDataRetirada() {
    return dataRetirada;
}

/**
 * Retorna a data prevista para devolução do livro.
 * @return data de devolução
 */
public Date getDataDevolucao() {
    return dataDevolucao;
}

/**
 * Retorna o status atual do empréstimo.
 * @return status do empréstimo
 */
public String getStatus() {
    return status;
}

// Setter para alterar o status do empréstimo (ex: mudar de "Ativo" para "Finalizado")

/**
 * Define um novo status para o empréstimo.
```



```
* @param status novo status do empréstimo
*/
public void setStatus(String status) {
    this.status = status;
}
}
```

### Explicação:

### Atributos:

- ``id``: Identificador único do empréstimo – **imutável**
- ``rgmAluno``: RGM do aluno que realizou o empréstimo
- ``idLivro``: ID do livro emprestado
- ``dataRetirada``: Data em que o livro foi retirado
- ``dataDevolucao``: Data prevista para devolução
- ``status``: Pode ser "Ativo", "Concluído" ou "Atrasado"

### Métodos:

- **Construtor**: Inicializa todos os atributos
- **Getters**: Permitem acesso a todos os atributos
- **Setter**: Apenas para ``status``, pois é o principal campo mutável durante o ciclo de vida do empréstimo

## Relacionamento entre as Classes

### 1. Aluno x Empréstimo:

- Um aluno (identificado pelo RGM) pode ter múltiplos empréstimos
- Relacionamento 1:N (um-para-muitos)

### 2. Livro x Empréstimo:

- Um livro pode estar associado a diferentes empréstimos ao longo do tempo
- Relacionamento 1:N (um-para-muitos)

### 3. Fluxo de Estados:

Quando um livro é emprestado:

- Status do livro muda para "Emprestado"
- Cria-se um novo Empréstimo com status "Ativo"

Quando o livro é devolvido:

- Status do livro volta para "Disponível"
- Status do Empréstimo muda para "Concluído"

## Princípios de Design Aplicados

### 1. Encapsulamento:

- Todos os atributos são privados
- Acesso controlado através de getters e setters limitados

### 2. Imutabilidade Seletiva:

- Campos de identificação (RGM, ID) não possuem setters
- Garante consistência dos dados críticos

### 3. Coesão:

- Cada classe tem responsabilidade bem definida
- Nenhuma classe faz mais do que deveria

### 4. Baixo Acoplamento:

- As classes não dependem diretamente umas das outras
- O acoplamento é feito através do sistema principal (Metodos.java)

Estas classes formam a base do modelo de dados do sistema e são utilizadas por todas as operações principais implementadas em Metodos.java.

## Validações e Tratamento de Erros

O sistema implementa diversas validações:

1. **Validação de RGM:** Deve ter exatamente 8 dígitos numéricos
2. **Validação de Email:** Usa regex para verificar formato válido
3. **Validação de Ano:** Não pode ser negativo ou maior que o ano atual + 1
4. **Transações:** Operações críticas usam transações para garantir integridade
5. **Tratamento de SQLException:** Todas as operações de banco são tratadas

## Considerações Finais

Este sistema foi projetado para:

- Ser intuitivo para usuários finais
- Manter a integridade dos dados através de validações e transações
- Ser modular e fácil de manter
- Fornecer relatórios úteis para gestão da biblioteca

Para expandir o sistema, pode-se considerar:

- Adicionar interface gráfica
- Implementar autenticação de usuários
- Adicionar mais relatórios estatísticos

- Implementar reserva de livros