

UNIVERSIDADE ANHEMBI MORUMBI
CAMPUS PAULISTA
PROGRAMAÇÃO DE SOLUÇÕES COMPUTACIONAIS

ATIVIDADE AVALIATIVA 3
BIBLIOTECA DIGITAL

Integrantes:

Eduarda Tobias Fernandes - RA: 12522216514;
Jennifer Reis Sicherolli De Souza - RA: 125221110547;
Katerine Linda Witkoski - RA: 12523131635;
Nathalie Mori Taylor Zampieri - RA: 1252224816;
Sérgio Silva Santos - RA: 12522221107;
Thifany Duarte Dos Santos - RA: 12522211469.

São Paulo/SP - 2023

1. Introdução

O Sistema da Biblioteca Digital é uma aplicação desenvolvida em Java para auxiliar na gestão de disponibilidade de livros e reserva de livros. O sistema permite o cadastro, consulta, atualização e exclusão de dados sobre usuários e livros.

Decidimos fornecer somente uma interface via terminal para interação dos usuários.

2. Desenvolvimento

O desenvolvimento do Sistema da Biblioteca Digital foi realizado seguindo os princípios da programação orientada a objetos, utilizando a linguagem de programação Java. O projeto foi estruturado em diferentes pacotes para separar as responsabilidades e facilitar a manutenção e extensibilidade do código. (*database, model, view*)

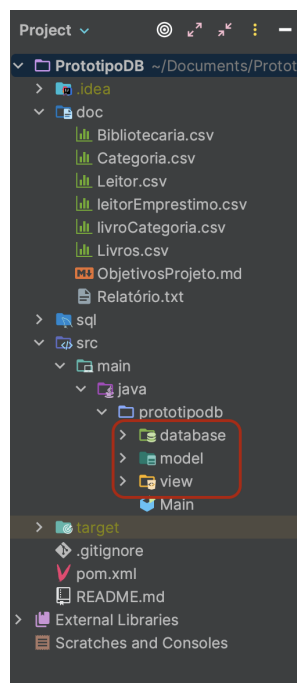


Figura 1: Estrutura do projeto

Foram utilizadas as seguintes tecnologias e ferramentas no desenvolvimento do sistema:

- MySQL: Utilizado como banco de dados para armazenar os dados da Biblioteca.
- JDBC: Utilizado para realizar a conexão com o banco de dados e executar consultas e atualizações.

2.2. Banco de Dados SQL:

'BIBLIOTECA' com 6 tabelas: (Relacional)

- Livros (contendo informações sobre 'cod_livro'; 'nome_livro'; 'autor_livro')
- Categoria (contendo informação sobre 'cod_categoria', 'nome_categoria')
- livrosCategoria (realizando um join entre as tabelas livros e categorias. Informações sobre 'cod_livro' e 'cod_categoria')
- Leitor (contendo informações sobre 'nome_leitor'; 'ra_leitor', 'email_leitor')
- leitorEmprestimo (contendo informações histórico de alugado/devolvido sobre 'codigo_livro'; 'status_livro'; 'ra_leitor')
- Bibliotecaria (contendo informações sobre 'nome_bibliotecaria'; 'cpf_bibliotecaria'; 'email_bibliotecaria')

2.3. Definições das classes:

- Usuário (classe pai, contendo informações de nome e e-mail das partes)
- Herança
- Leitor (classe filha, contendo informações da classe pai e um atributo exclusivo "RA")
- Bibliotecária (classe filha, contendo informações da classe pai e um atributo exclusivo "CPF")
- Categoria (Como existe uma relação n:n com livros decidimos criar uma classe para categorias)

- Empréstimo (Contendo a lógica de ``reservar()`` e ``devolver()`` para os livros)
- Livro (contendo toda a parte de gestão dos livros)
- Database (conexão com o banco de dados)

2.4. Decisões de funcionalidades:

O Sistema de Gerenciamento de Funcionários possui as seguintes funcionalidades principais:

1. Usuários:

- Permite cadastrar novos leitores e bibliotecários: ``criarLeitor()`` e ``criarBibliotecaria()``.
- Permite alterar leitores e bibliotecários: ``alterarLeitor()`` e ``alterarBibliotecaria()``.
- Permite deletar leitores e bibliotecários: ``deletarLeitor()`` e ``deletarBibliotecaria()``.

2. Categorias:

- Permite cadastrar uma nova categoria: ``criarCategoria()``.
- Permite alterar uma categoria: ``alterarCategoria()``.
- Permite visualizar todas as categorias em uma tabela no terminal: ``lerCategorias()`` e ``mostrarCategorias()``.

3. Livros:

- Permite cadastrar um novo livro: ``criarLivro()``.
- Permite alterar cadastro de um livro: ``alterarLivro()``.
- Permite deletar livro caso esteja danificado: ``deletarLivro()``.
- Permite visualizar todos os livros em uma tabela no terminal: ``lerLivros()`` e ``mostrarLivros()``.
- Permite filtrar busca de usuário por título de livro: ``filtroLivrosTitulo()`` e ``mostrarFiltroPorTitulo()``.
- Permite filtrar busca de usuário por categoria: ``filtroLivrosCategoria()`` e ``mostrarFiltroPorCategoria()``.

3. Emprestimo:

- Permite que um leitor reserve um livro: ``reservarLivro()`` e ``verificarDisponibilidadeLivro()``.
- Permite que um leitor devolva um livro: ``devolverLivro()`` e ``verificarLivroEmprestado()``.
- Permite visualizar todos os empréstimos em uma tabela no terminal: ``lerEmprestimo()`` e ``mostrarEmprestimos()``.

4. Database:

- Permite uma conexão com o banco de dados MySQL: ``Database()``.

2.5. Descrição do código-fonte

Para representar uma das soluções desenvolvidas, de modo a abordar a estrutura de dados utilizada e as classes para gerenciamento e interação com o projeto. Seguimos com o exemplo de código da classe 'Leitor', na qual possui atributos Nome (String), RA (String) e e-mail (String). Ela é uma extensão da classe pai 'Usuario', contendo informação de RA como exclusiva da classe Leitor.

Nessa primeira representação temos a classe conceito, contendo o método construtor utilizando *super(nome, email)* para referenciar os atributos da classe pai e o *getRA()* e *setRA()*:

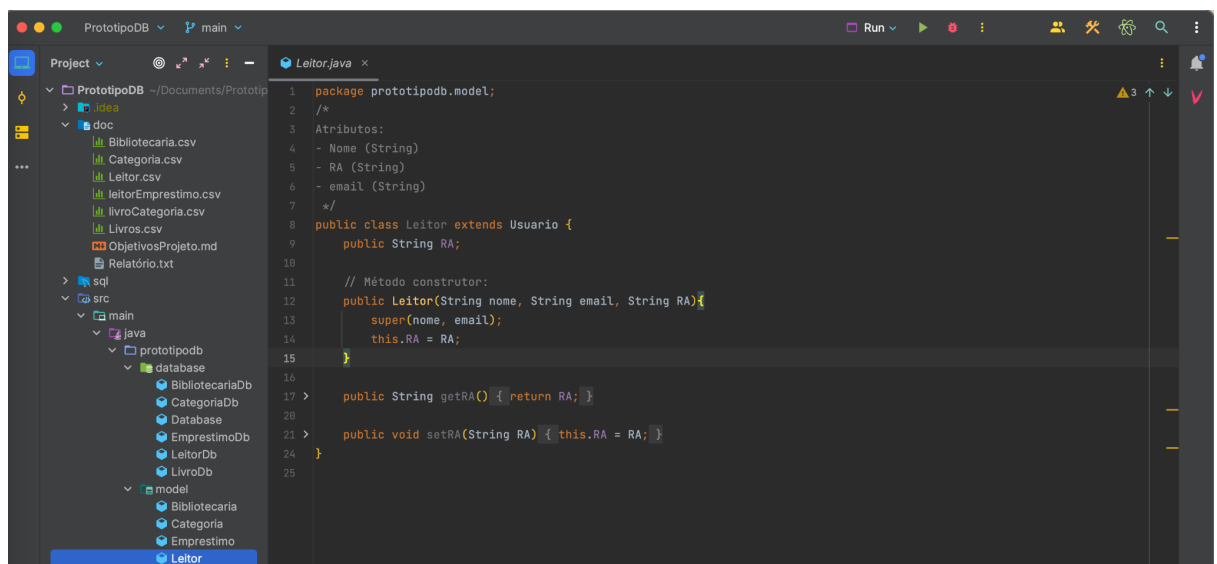


Figura 2: Classe Leitor - package model

Nessa segunda representação temos a classe Leitor com métodos responsáveis pelas interações diretas com o banco de dados. Na qual, necessariamente é preciso fazer uma conexão com MySQL.

Foi baseado nas operações básicas do desenvolvimento de uma aplicação CRUD (Create, Read, Update and Delete).

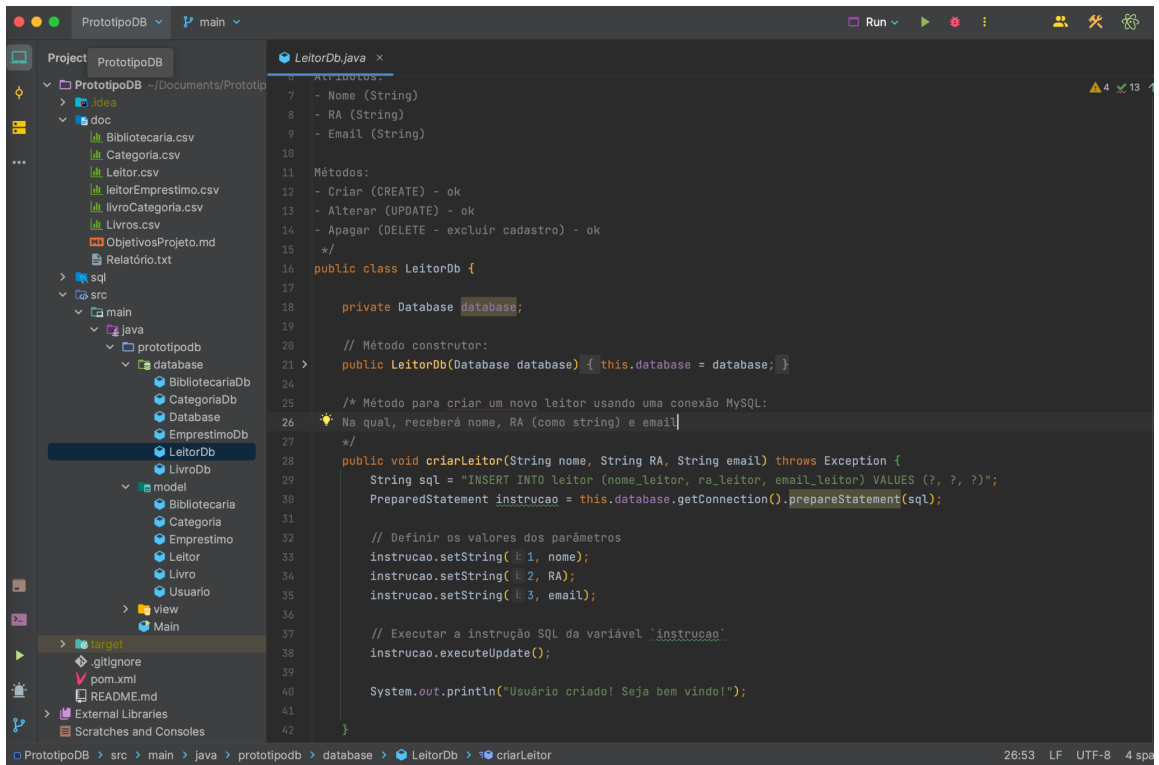


Figura 3: Classe Leitor - package database 1

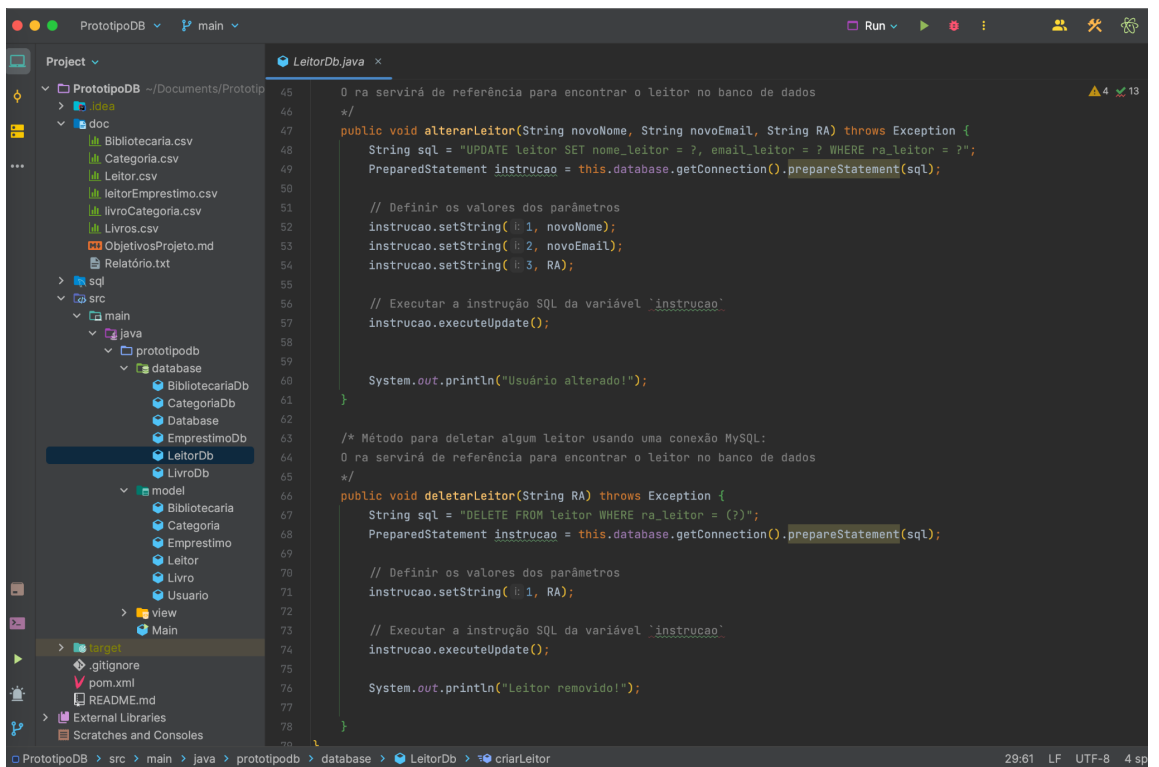
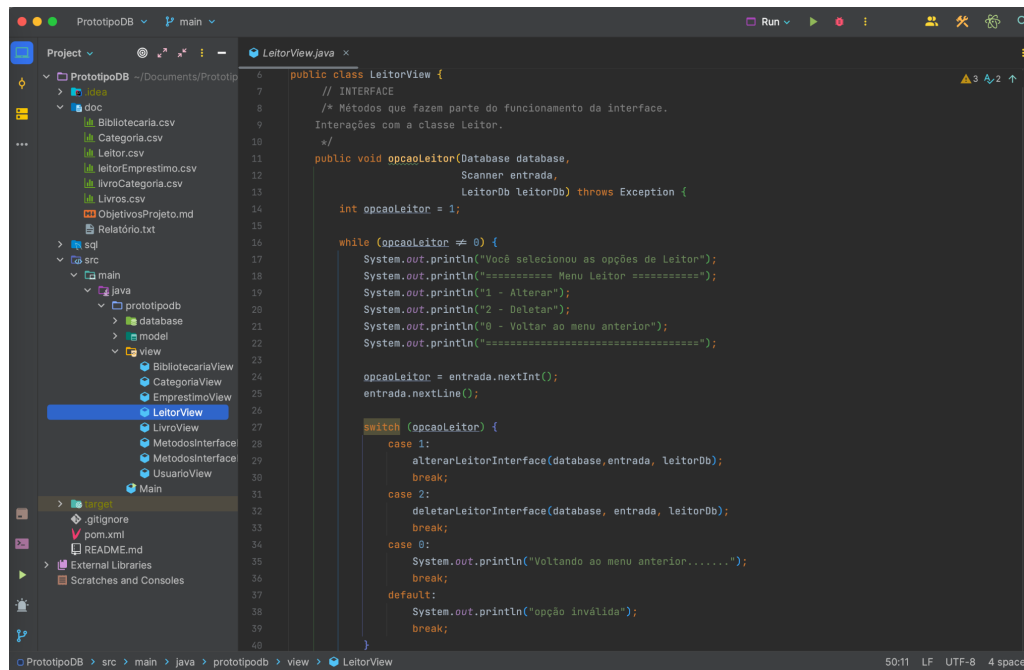


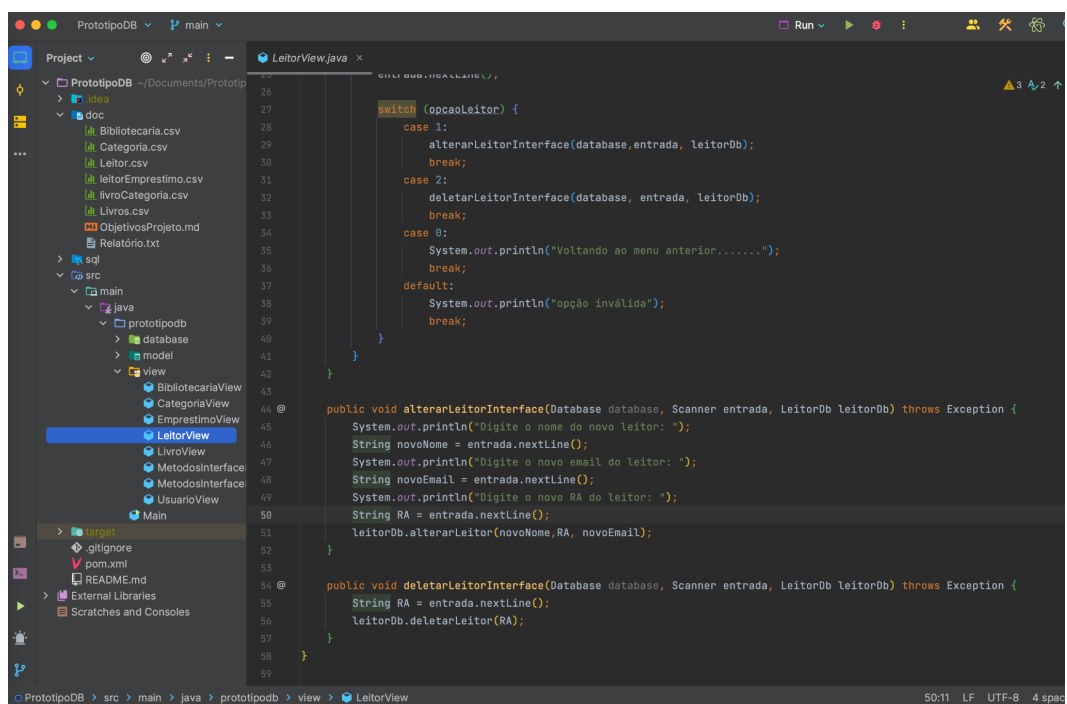
Figura 4: Classe Leitor - package database 2

Nessa terceira representação temos a classe Leitor com métodos responsáveis pela visualização do sistema, métodos que contém como retorno qualquer saída de terminal.



```
6 public class LeitorView {
7     // INTERFACE
8     /* Métodos que fazem parte do funcionamento da interface.
9     Interações com a classe Leitor.
10    */
11    public void opcaoLeitor(Database database,
12                          Scanner entrada,
13                          LeitorDb leitorDb) throws Exception {
14        int opcaoLeitor = 1;
15
16        while (opcaoLeitor != 0) {
17            System.out.println("Você selecionou as opções de Leitor");
18            System.out.println("===== Menu Leitor =====");
19            System.out.println("1 - Alterar");
20            System.out.println("2 - Deletar");
21            System.out.println("0 - Voltar ao menu anterior");
22            System.out.println("=====");
23
24            opcaoLeitor = entrada.nextInt();
25            entrada.nextLine();
26
27            switch (opcaoLeitor) {
28                case 1:
29                    alterarLeitorInterface(database, entrada, leitorDb);
30                    break;
31                case 2:
32                    deletarLeitorInterface(database, entrada, leitorDb);
33                    break;
34                case 0:
35                    System.out.println("Voltando ao menu anterior.....");
36                    break;
37                default:
38                    System.out.println("opção inválida");
39                    break;
40            }
41        }
42    }
43 }
```

Figura 5: Classe Leitor - package view 1



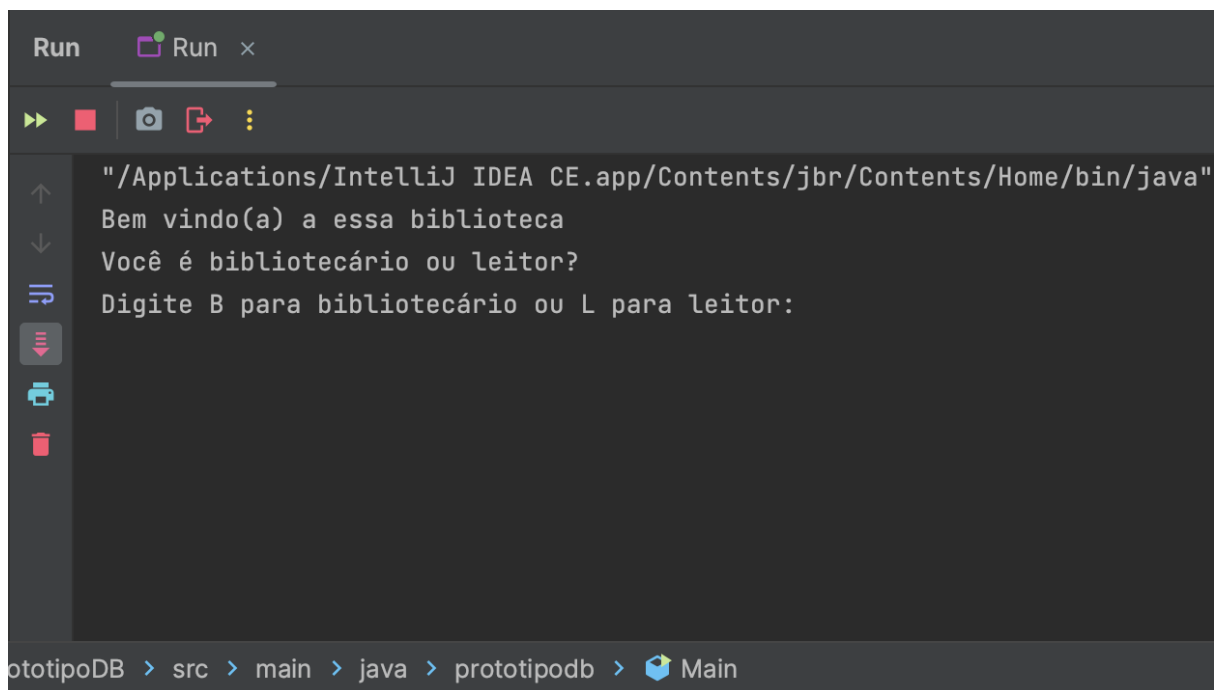
```
26
27
28 switch (opcaoLeitor) {
29     case 1:
30         alterarLeitorInterface(database, entrada, leitorDb);
31         break;
32     case 2:
33         deletarLeitorInterface(database, entrada, leitorDb);
34         break;
35     case 0:
36         System.out.println("Voltando ao menu anterior.....");
37         break;
38     default:
39         System.out.println("opção inválida");
40         break;
41 }
42 }
43
44 @
45 public void alterarLeitorInterface(Database database, Scanner entrada, LeitorDb leitorDb) throws Exception {
46     System.out.println("Digite o nome do novo leitor: ");
47     String novoNome = entrada.nextLine();
48     System.out.println("Digite o novo email do leitor: ");
49     String novoEmail = entrada.nextLine();
50     System.out.println("Digite o novo RA do leitor: ");
51     String RA = entrada.nextLine();
52     leitorDb.alterarLeitor(novoNome, RA, novoEmail);
53 }
54 @
55 public void deletarLeitorInterface(Database database, Scanner entrada, LeitorDb leitorDb) throws Exception {
56     String RA = entrada.nextLine();
57     leitorDb.deletarLeitor(RA);
58 }
59 }
```

Figura 6: Classe Leitor - package view 2

Essa foi apenas uma breve explicação do código fonte relevante da classe 'Leitor'. A aplicação completa depende das outras classes, já citadas no relatório.

2.6. Criação da interface

Para visualização do sistema da biblioteca escolhemos fazer a interface por meio do terminal.



```
Run Run x
"/Applications/IntelliJ IDEA CE.app/Contents/jbr/Contents/Home/bin/java"
Bem vindo(a) a essa biblioteca
Você é bibliotecário ou leitor?
Digite B para bibliotecário ou L para leitor:
```

prototipoDB > src > main > java > prototipodb > Main

Foi separado toda a lógica de funcionamento das opções possíveis de interação com o usuário por métodos.

Construídos em classes:

- MetodosInterfaceB: interações da Bibliotecária no sistema
- MetodosInterfaceL: interações do Leitor no sistema

3. Uso

Exemplo de chamada: (Criando um novo cadastro para Leitor)

```
// Informações do BD
String username = "root";
String password = "rootroot";
String databaseName = "BIBLIOTECA";
String serverName = "localhost";

// Criando uma nova conexão
Database database = new Database(serverName, databaseName, password, username);
// Criando um cadastro de leitor
LeitorDb leitorDb = new LeitorDb(database);
leitorDb.criarLeitor("Katerine Witkoski", "12344559", "katerinewitkoski@gmail.com")
```

4. Estrutura do Projeto

Existe uma pasta `doc` (contendo os objetivos do trabalho e esse mesmo relatório em .pdf) e uma pasta `sql` (contendo todos os scripts relacionados ao banco de dados).

Para utilizar o sistema compile o arquivo `Main`.

Todo o projeto foi desenvolvido com versão de controle git/github, disponível em *prototipoBD*.¹

¹ <https://github.com/katerine-dev/prototipoBD>