****

**Trabalho de Programação Orientada à Objetos**

**Sistema Para Gerenciamento de Biblioteca - OBLibrary**

**Professor: Saulo Henrique Cabral Silva**

**Alunos: Lucas Cardoso Costa**

**Ouro Branco**

**2025**

b

Introdução

Este documento apresenta a documentação do trabalho desenvolvido para a disciplina de Programação Orientada a Objetos, do curso de Bacharelado em Sistemas de Informação, oferecida pelo Instituto Federal de Minas Gerais (IFMG) - Campus Ouro Branco. A atividade foi proposta pelo professor Saulo Henrique Cabral Silva e consiste na implementação de um sistema de gestão para a Biblioteca Municipal de Ouro Branco. O código desenvolvido está disponível no endereço [https://github.com/brazillucas/POO22024/](https://github.com/brazillucas/POO22024/tree/main/OBLibrary).

O objetivo deste trabalho é desenvolver um sistema de gestão de biblioteca funcional, interativo e baseado nos conceitos fundamentais da Programação Orientada a Objetos (POO), como herança, encapsulamento e polimorfismo. O sistema foi desenvolvido para ser capaz de gerenciar as operações da biblioteca. O sistema permite o cadastro de usuários (alunos, professores e bibliotecários), o gerenciamento de obras literárias (títulos, autores e disponibilidade) e a realização de operações como empréstimos e devoluções. Todos os dados são persistidos em arquivos **.txt** para garantir que o estado do sistema seja mantido mesmo após o encerramento do programa. Além disso, o sistema gera relatórios sobre obras emprestadas e usuários com atrasos, facilitando o acompanhamento das atividades da biblioteca.

Implementação

Abaixo serão listadas as classes implementadas no sistema na ordem em que foram desenvolvidas e adicionadas ao projeto. Nem todos os métodos serão citados, apenas os mais relevantes e usuais.

### 1. Classe ObraLiteraria

**Responsabilidade: Representa uma obra literária no sistema, armazenando informações como título, autor, quantidade disponível e total.**

**Atributos:**

* **id (int):** Identificador único da obra.
* **titulo (String):** Título da obra.
* **autor (String):** Autor da obra.
* **quantidadeDisponivel (int):** Quantidade de exemplares disponíveis para empréstimo.
* **quantidadeTotal (int):** Quantidade total de exemplares da obra.

**Métodos:**

* **consultarInformacoes():** Retorna uma string com as informações completas da obra*(ID, título, autor, quantidade disponível e total).*
* **getQuantidadeDisponivel():** Retorna a quantidade de exemplares disponíveis da obra.
* **atualizarQuantidade(int quantidade):** Atualiza a quantidade disponível da obra (incrementa ou decrementa).
* **getId():** Retorna o ID da obra.
* **getTitulo():** Retorna o título da obra.
* **getQuantidadeTotal():** Retorna a quantidade total de exemplares da obra.

### 2. Classe Usuario

**Responsabilidade:** A classe abstrata que representa um usuário de onde os outros deveram herdar, armazenando informações como **nome**, **email**, **senha**, **tipo de usuário**, **bloqueio** e **empréstimos realizados**.

**Atributos:**

* **nome (String):** Nome do usuário.
* **email (String):** Email do usuário.
* **senha (String):** Senha do usuário.
* **TipoUsuario (int):** Tipo do usuário (*1 - Aluno, 2 - Professor, 3 - Bibliotecário*).
* **emprestimosRealizados (List<Emprestimo>):** Lista de empréstimos realizados pelo usuário.
* **bloqueado (boolean):** Indica se o usuário está bloqueado por atraso.

**Métodos:**

* **login(String email, String senha):** Verifica as credenciais do usuário e retorna true se forem válidas.
* **verificarBloqueio():** Retorna true se o usuário estiver bloqueado por atraso na devolução de algum livro.
* **validacaoEmprestimo(ObraLiteraria obra):** Valida se o usuário pode realizar um empréstimo da obra, considerando bloqueios, limite de empréstimos e disponibilidade da obra.
* **verificarLimiteEmprestimos():** Verifica se o usuário atingiu o limite de empréstimos permitidos com base no tipo de usuário.
* **getNome(): Retorna o nome do usuário.**
* **getEmail(): Retorna o email do usuário.**
* **getSenha():** Retorna a senha do usuário. Foi escolhido não trabalhar com encriptação, por não ser um requisito do trabalho e pelo tempo para implementação.
* **getTipoUsuario():** Retorna o tipo do usuário.
* **getEmprestimosRealizados():** Retorna a lista de empréstimos realizados pelo usuário.
* **exibirObrasEmprestadas():** Exibe as obras atualmente emprestadas pelo usuário.

### 3. Classe Aluno

**Responsabilidade:** Representa um aluno no sistema, estendendo a classe *Usuario* e adicionando informações específicas como matrícula e curso.

**Atributos:**

* **matricula (String):** Número de matrícula do aluno.
* **curso (String):** Curso ao qual o aluno está vinculado.

**Métodos:**

* **getMatricula():** Retorna a matrícula do aluno.
* **getCurso():** Retorna o curso do aluno.

### 4. Classe Professor

**Responsabilidade:** Representa um professor no sistema, estendendo a classe *Usuario* e adicionando informações específicas como departamento.

**Atributos:**

* **departamento (String):** Departamento ao qual o professor está vinculado.

**Métodos:**

* **getDepartamento():** Retorna o departamento do professor.

**5. Classe Bibliotecario**

**Responsabilidade:** Representa um bibliotecário no sistema, estendendo a classe *Usuario* e adicionando funcionalidades específicas como cadastro de usuários e geração de relatórios.

**Atributos:**

* **telefone (String):** Número de telefone do bibliotecário.
* **devolucoesRealizadas (int):** Quantidade de devoluções realizadas pelo bibliotecário.

**Métodos:**

* **getDevolucoesRealizadas():** Retorna a quantidade de devoluções realizadas pelo bibliotecário.
* **incrementaDevolucoesRealizadas():** Incrementa o contador de devoluções realizadas pelo bibliotecário.
* **cadastrarUsuario(Biblioteca biblioteca):** Cadastra um novo usuário no sistema (aluno, professor ou bibliotecário) e salva os dados no arquivo.
* **gerarRelatorios(Biblioteca biblioteca):** Gera relatórios de obras emprestadas e usuários com atraso, se houver.

### 6. Classe Emprestimo

**Responsabilidade:** Representa um empréstimo no sistema, armazenando informações sobre o usuário, a obra, datas de empréstimo, devolução prevista e realizada, além do status de atraso e ativo.

**Atributos:**

* **usuario (Usuario):** Usuário que realizou o empréstimo.
* **obra (ObraLiteraria):** Obra literária emprestada.
* **dataEmprestimo (LocalDate):** Data em que o empréstimo foi realizado.
* **dataDevolucaoPrevista (LocalDate):** Data prevista para a devolução da obra.
* **dataDevolucaoRealizada (LocalDate):** Data em que a obra foi devolvida (se aplicável).
* **emprestimoAtivo (boolean):** Indica se o empréstimo ainda está ativo (ou seja, sem uma data de devolução).
* **atrasado (boolean):** Indica se o empréstimo está em atraso.

**Métodos:**

* **verificarAtraso():** Verifica se o empréstimo está em atraso com base na data atual e na data de devolução prevista.
* **getObra():** Retorna a obra literária emprestada.
* **getUsuario():** Retorna o usuário que realizou o empréstimo.
* **getDataEmprestimo():** Retorna a data em que o empréstimo foi realizado.
* **getAtrasado():** Retorna true se o empréstimo estiver em atraso.
* **getDataDevolucaoPrevista():** Retorna a data prevista para a devolução da obra.
* **setDevolucaoRealizada():** Registra a devolução da obra, atualizando a *data de devolução realizada*, o *status de atraso* e o *status de empréstimo ativo*.
* **getDataDevolucaoRealizada():** Retorna a data em que a obra foi devolvida.
* **setDataDevolucaoPrevista(LocalDate dataDevolucaoPrevista):** Define a data prevista para a devolução.
* **getEmprestimoAtivo():** Retorna true se o empréstimo ainda estiver ativo**.**
* **getDiasAtraso():** Calcula e retorna a quantidade de dias de atraso no empréstimo.
* **toString():** Retorna uma representação textual do empréstimo, incluindo informações como *usuário*, *obra*, *datas* e *status de atraso*.

### 7. Classe Biblioteca

**Responsabilidade:** Gerencia todas as operações do sistema e coordena as interações entre as entidades.

**Atributos:**

* **usuarios (List<Usuario>):** Lista de todos os usuários cadastrados.
* **usuarioLogado (Usuario):** Usuário atualmente logado no sistema.
* **obras (List<ObraLiteraria>):** Lista de todas as obras disponíveis.
* **emprestimos (List<Emprestimo>):** Lista de todos os empréstimos realizados.

**Métodos:**

* **carregarDados():** Carrega os dados do sistema (usuários, obras e empréstimos) a partir de arquivos **.txt**.
* **loginUsuario(String email, String senha):** Verifica as credenciais e retorna o usuário logado, se válido.
* **logoutUsuario():** Encerra a sessão do usuário atual.
* **cadastrarUsuario(Usuario usuario):** Cadastra um novo usuário no sistema (operação exclusiva do bibliotecário).
* **gerarRelatorioObrasEmprestadas():** Gera um relatório das obras atualmente emprestadas.
* **gerarRelatorioUsuariosComAtraso():** Gera um relatório dos usuários com empréstimos em atraso.
* **realizarEmprestimo(ObraLiteraria obra, Usuario usuario):** Realiza o empréstimo de uma obra para um usuário.
* **realizarDevolucao(ObraLiteraria obra, Usuario usuario):** Registra a devolução de uma obra por um usuário.
* **exibirObrasDisponiveis():** Exibe as obras disponíveis no acervo.
* **consultarObra(String titulo):** Consulta uma obra pelo título.
* **consultarObra(int id):** Consulta uma obra pelo ID.
* **selecionarObra():** Permite ao usuário selecionar uma obra disponível para empréstimo.
* **buscarObra(int id):** Busca uma obra pelo ID.
* **getEmprestimosAtivos():** Verifica se há empréstimos ativos no sistema.
* **getUsuariosComAtraso():** Verifica se há usuários com empréstimos em atraso.
* **executarConsulta():** Executa uma consulta de obra pelo ID ou título.
* **fechamentoSistema():** Salva os dados do sistema antes de encerrar a execução chamando as funções *salvarEmprestimosArquivo* e *salvarUsuariosArquivo*.
* **salvarEmprestimosArquivo():** Salva os dados dos empréstimos em um arquivo **.txt**.
* **salvarUsuariosArquivo():** Salva os dados dos usuários em um arquivo **.txt**.

### 8. Classe Main

**Responsabilidade:** Representa o ponto de partida do sistema, inicializando a biblioteca, carregando dados e gerenciando o loop principal do programa.

**Atributos:**

* **Nenhum atributo declarado na classe.**

**Métodos:**

* **main(String[] args):** Método principal que inicia a execução do sistema. Realiza as seguintes operações:
  + Cria uma instância da classe **Biblioteca**.
  + Carrega os dados do sistema (*usuários, obras e empréstimos*) a partir de arquivos.
  + Exibe o menu principal e processa as opções escolhidas pelo usuário.
  + Encerra o sistema salvando os dados quando o usuário escolhe sair.

### 9. Classe Menu

**Responsabilidade:** Gerencia a exibição dos menus do sistema para diferentes tipos de usuários (*aluno, professor e bibliotecário*).

**Atributos:**

* **Nenhum atributo declarado na classe.**

**Métodos:**

* **exibirMenuPrincipal():** Exibe o menu principal do sistema com opções para *login* ou *sair*.
* **exibirMenuUsuario(Usuario usuario):** Exibe o menu específico para alunos e professores, incluindo opções como *realizar empréstimo*, *verificar empréstimos ativos*, *verificar bloqueio* e *consultar obras*.
* **exibirMenuBibliotecario(Usuario usuario):** Exibe o menu específico para bibliotecários, incluindo opções como *cadastrar usuário*, *registrar devolução*, *gerar relatórios* e *consultar obras*.

10. **Classe Entrada**

**Responsabilidade:** Gerencia a interação com o usuário, validando entradas de dados, solicitando senhas, emails e realizando operações de limpeza de tela e pausa.

**Atributos:**

* **Nenhum atributo declarado na classe.**

**Métodos:**

* **solicitarEntradaValida(String mensagem, String regex, String mensagemErro):** Solicita uma entrada do usuário e valida-a com base em uma expressão regular (**regex**). Exibe uma mensagem de erro caso a entrada seja inválida.
* **solicitarSenha():** Solicita uma senha do usuário, validando-a com base em um padrão específico (4-8 caracteres, incluindo letras maiúsculas, minúsculas, números e caracteres especiais) e oculta os caracteres digitados pelo usuário.
* **solicitarEmail():** Solicita um e-mail do usuário, validando-o com base em um padrão de formato de e-mail.
* **limparTela():** Limpa a tela do console.
* **aguardarEnter():** Aguarda que o usuário pressione Enter para continuar.
* **finalizarFuncao():** Combina as funcionalidades de **aguardarEnter** e **limparTela**, permitindo uma pausa e limpeza da tela após uma operação.

Por fim, temos o diagrama de classes do projeto:

|  |
| --- |
|  |
| Figura 1 - Diagrama de Classes Simplificado |

### Conclusão

O desenvolvimento deste trabalho levantou diversos desafios na sua implementação, principalmente pela utilização de métodos ainda pouco treinados, mas com toda certeza contribuiu em muito para aprimorar meus conhecimentos em programação orientada a objetos.

### Referências

[Using Pattern Matching - Dev.java](https://dev.java/learn/pattern-matching/)

[How to hide harded coded password in Java source code](https://stackoverflow.com/questions/25217341/how-to-hide-harded-coded-password-in-java-source-code)