IPV – Instituto Politécnico de Viseu

ESTGV – Escola Superior de Tecnologia e Gestão de Viseu

Departamento de Informática

Relatório do Projeto

Engenharia Informática

Unidade Curricular: Programação Orientada a Objetos

Realizado por:

José Arrais, pv23747

Julino Mendonça, pv22529

João Cruz, pv25178

Rodrigo Pereira, pv27450

Viseu, 2024

Índice

Índice de imagens

Introdução

O presente trabalho surge no âmbito da disciplina de Programação orientada a objetos, do curso de Engenharia Informática, da Escola Superior de Tecnologia e Gestão de Viseu. O objetivo do trabalho prático é aplicar conhecimentos adquiridos em aulas, além de aprimorar o domínio da linguagem C++, criando um sistema de gestão de uma biblioteca que utilize extensivamente os conceitos de herança e polimorfismo para modelar diversos tipos de livros, leitores e funcionalidades.

Bibliotecas utilizadas

O projeto utiliza diversas bibliotecas, tanto da biblioteca padrão do C++ quanto bibliotecas personalizadas, para implementar suas funcionalidades. Entre as bibliotecas padrão, destacam-se <iostream>, utilizada para entrada e saída de dados, como leitura com cin e escrita com cout, e <windows.h>, que oferece funcionalidades específicas para o sistema operacional Windows, como manipulação da saída de texto utilizando palavras acentuadas em português. Além disso, são utilizadas <list> e <map> para manipulação de estruturas de dados dinâmicas e associativas, respectivamente, enquanto <fstream> permite a leitura e escrita em arquivos. A biblioteca <string> facilita a manipulação de cadeias de caracteres. Por fim, <vector> é utilizada para trabalhar com vetores dinâmicos, permitindo o armazenamento e a manipulação eficiente de sequências de elementos.

O projeto também conta com bibliotecas personalizadas, como "Biblioteca.h", que é a principal do projeto, agregando funcionalidades relacionadas à gestão de livros, pessoas e empréstimos. Outras bibliotecas personalizadas incluem "menu.h", que gere a criação de menus interativos, “Uteis.h", que fornece funções auxiliares diversas, essenciais para a operação geral do sistema. "Pessoa.h" e "Geral.h" estruturam os dados e comportamentos associados a pessoas e livros, enquanto "Emprestimos.h" centraliza as operações relacionadas a empréstimos, integrando diferentes partes do sistema. A organização modular dessas bibliotecas promove a reutilização, facilita a manutenção do código e contribui para a escalabilidade do projeto.

Interface do utilizador

A interface do utilizador é a maneira pela qual um utilizador interage com o sistema, estabelecendo uma relação entre a interação e o funcionamento da aplicação. A interface é fundamental para garantir que a experiência seja agradável e satisfatória, atendendo às necessidades do utilizador e melhorando a experiência geral.

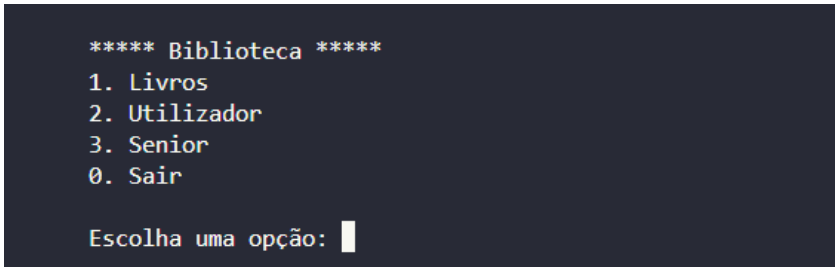

Figura 1 - - Interface menu principal

Main.cpp

A função main é o ponto de entrada do programa e organiza sua execução. Ela configura a codificação UTF-8 no console para suportar caracteres especiais, inicializa os objetos Biblioteca e Uteis, e chama a função LerGeral para carregar dados essenciais. Em seguida, executa o MENU_PRINCIPAL, que gere a interação com o utilizador e as funcionalidades do sistema. Por fim, retorna 0 para indicar o encerramento bem-sucedido do programa.

```
int main() {
    SetConsoleOutputCP(CP_UTF8);
    Biblioteca biblioteca;
    Uteis uteis;
    uteis.LerGeral(biblioteca);
    MENU_PRINCIPAL(biblioteca,uteis);
    return 0;
}
```

Figura 2 - função main

Leitura dos Ficheiros

Para a leitura dos ficheiros ao iniciar o programa foi utilizada a função LerGeral() que é responsável por realizar a leitura inicial de dados essenciais para o funcionamento do sistema, permitindo ao utilizador escolher o formato do arquivo a ser utilizado (.txt ou .csv). Essa escolha influencia como as informações serão processadas e carregadas para as classes relevantes.
A partir da extensão selecionada, a função chama diversos métodos para ler informações específicas:

- LerLimitesPorCategoria: Lê dados sobre os limites de livros que uma pessoa pode pedir ao mesmo tempo, dependendo da sua categoria.

- LerDesconto: Lê informações sobre descontos aplicáveis a cada categoria.

- LerLivrosPorCategoria: Carrega os dados de livros e os organiza por categoria, integrando-os na lista de livros organizados por uma chave(a categoria) na Biblioteca.

- LerPessoasPorCategoria: Lê informações sobre pessoas, também organizando-as por categoria no objeto Biblioteca.

- Métodos da classe Biblioteca como LerEmprestimos e LerReservas são chamados para carregar dados de empréstimos e reservas.

data > limitesPorCategoria.csv > data

```
1  Estudante ; 8
2  LeitorComum ; 3
3  Professor ; 10
4  Senior ; 10
```

Figura 3 - exemplo de um ficheiro

```
bool Uteis::LerLimitesPorCategoria(string extensaoArquivo) {
    ifstream file("./data/limitesPorCategoria"+extensaoArquivo);
    if (!file.is_open()) {
        cerr << "Erro ao abrir o arquivo: limitesPorCategoria.txt" << endl;
        system("pause");
        return false;
    }

    string linha;
    while (getline(file, linha)) {
        stringstream ss(linha);
        string categoria;
        int limite;

        getline(ss, categoria, ';');
        ajustarCaracteres(categoria);

        string Strlimite;
        getline(ss, Strlimite, ';');
        limite = stoi(Strlimite);

        addLimiteLivros(categoria,limite);
    }

    file.close();
    return true;
}
```

Figura 4 - exemplo de uma função para leitura de ficheiro

Biblioteca.h

A classe Biblioteca utiliza listas encadeadas para organizar dados de livros, leitores, empréstimos e reservas de forma categorizada. Essa abordagem permite flexibilidade na manipulação e busca, além de suportar operações como adicionar, listar e buscar informações com eficiência. As listas são integradas aos mapas para facilitar o acesso por categoria, garantindo uma gestão estruturada e modular dos dados do sistema.

- livrosPorCategoria: Armazena listas de ponteiros para objetos da classe Geral, representando livros, organizados por categorias.

- emprestimosPorCategoria: Contém listas de objetos Emprestimo, separadas por categorias, permitindo o controle detalhado de empréstimos.

- reservasPorCategoria: Gere listas de objetos Emprestimo que representam reservas, também organizadas por categorias.

- leitores: Armazena listas de ponteiros para objetos da classe Pessoa, categorizando os leitores de acordo com critérios específicos.

```
class Biblioteca {
private:
    // Armazena os livros organizados por categoria
    map<string, list<Geral*>> livrosPorCategoria;
    map<string, list<Emprestimo>> emprestimosPorCategoria;
    map<string, list<Emprestimo>> reservasPorCategoria;
    map<string, list<Pessoa*>> leitores;
```

Figura 5 - Listas encadeas do programa

Principais métodos dessa classe

1. Destrutor da classe Biblioteca: Biblioteca::~Biblioteca() é responsável por liberar a memória alocada dinamicamente e limpar as estruturas de dados utilizadas, garantindo que não ocorram vazamentos de memória quando o objeto Biblioteca é destruído.

   Funcionamento:

   - Liberação dos Livros (livrosPorCategoria):
     Percorre cada categoria no mapa, para cada livro armazenado na lista, chama delete para liberar a memória ocupada. Após liberar os ponteiros, limpa a lista (clear()) e remove a categoria do mapa.

   - Liberação dos Leitores (leitores):
     Segue a mesma lógica dos livros. Cada ponteiro para objetos Pessoa é deletado, as listas de leitores são esvaziadas, e o mapa é limpo.

   - Limpeza de Empréstimos (emprestimosPorCategoria):
     As listas de empréstimos em cada categoria são esvaziadas com o clear(), e o mapa é limpo. Como os objetos armazenados não são ponteiros, não é necessário usar o delete.

   - Limpeza de Reservas (reservasPorCategoria):
     Também limpa as listas de reservas e o mapa, similar à lógica de empréstimos.

```cpp
Biblioteca::~Biblioteca() {
// Liberar livros
    for (auto& categoria : livrosPorCategoria) {
        for (auto livro : categoria.second) {
            delete livro;
        }
        categoria.second.clear();
    }
    livrosPorCategoria.clear();

    // Liberar leitores
    for (auto& categoriaL : leitores) {
        for (auto leitor : categoriaL.second) {
            delete leitor;
        }
        categoriaL.second.clear();
    }
    leitores.clear();


     for (auto& categoriaE : emprestimosPorCategoria) {
        categoriaE.second.clear(); // limpar a lista de empréstimos
    }
    emprestimosPorCategoria.clear();

    // Limpar reservas
    for (auto& categoriaR : reservasPorCategoria) {
        categoriaR.second.clear(); // limpar a lista de reservas
    }
    reservasPorCategoria.clear();

    cout << "Biblioteca destruída com sucesso.\n";
}
```

Figura 6 - Destrutor da Biblioteca

2. Gestão de Livros: Métodos como adicionarLivro, removerLivro e listarLivrosDisponiveis manipulam as listas de livros, facilitando a organização e o acesso.

3. Gestão de Leitores: Funções como adicionarLeitor e listarLeitores permitem incluir e listar leitores de categorias específicas.

4. Empréstimos e Reservas: Métodos como registrarEmprestimo, registrarReserva e transformarReservaEmEmprestimo operam sobre as listas de empréstimos e reservas, organizando o fluxo desses processos.

Geral.h

A classe Geral é uma classe base abstrata que serve como modelo para representar livros no sistema. Ela fornece os atributos e métodos fundamentais, além de estabelecer a interface que as subclasses devem implementar, garantindo consistência e flexibilidade na manipulação de diferentes tipos de livros.

Atributos

- título: Armazena o título do livro.
- autor: Guarda o nome do autor do livro.
- ano: Representa o ano de publicação.
- disponivel: Indica se o livro está disponível para empréstimo ou reserva.
- categoria: Define a categoria à qual o livro pertence.

```
class Geral
{
    protected:
    string titulo;
    string autor;
    int ano
    bool di   std::__cxx11::string Geral::categoria
    string categoria;

public:
    Geral(string categoria,string titulo, string autor, int ano, bool disponivel);
    virtual ~Geral();
```

Figura 7 - classe Geral.h

Métodos

1. Construtor e Destrutor:
    - O construtor inicializa os atributos principais do livro.
    - O destrutor é virtual, permitindo que as subclasses limpem recursos adequadamente.

2. Métodos Virtuais Puros:
    - descricao(): É implementado pelas subclasses para exibir uma descrição detalhada do livro.
    - getCodigo(): Retorna o código único do livro, cuja lógica depende da implementação na subclasse, sendo ele ISBN se for livro ou ISSN se for Jornal ou Revista.
    - AlterarInformacaoDoLivro(): Permite modificar os detalhes do livro, definida em subclasses.

3. Método Virtual:
    - escreverFicheiro(ofstream& file): Define a lógica para gravar os dados do livro em um arquivo. Pode ser sobrescrito nas subclasses para adicionar detalhes específicos.
4. Métodos Gerais:
    - getCategoria() e getTitulo(): Retornam, respectivamente, a categoria e o título do livro.
    - isDisponivel(): Verifica se o livro está disponível.
    - setDisponivel() e setInDisponivel(): Alteram o estado de disponibilidade do livro.

```
virtual void descricao() const = 0;
virtual string getCodigo() const = 0;
virtual bool escreverFicheiro(ofstream& file);

string getCategoria();
string getTitulo();

bool isDisponivel();
void setDisponivel();
void setInDisponivel();

virtual void AlterarInformacaoDoLivro() = 0;
```

Figura 8 - Métodos da classe Geral

Exemplo de uma subclasse de Geral

A classe LivroEducativo é uma extensão da classe Geral, projetada para representar livros voltados ao uso educacional. Ela adiciona atributos específicos, como ISBN, nível educacional e matéria, para enriquecer a descrição e a gestão desse tipo de livro. Os métodos sobrescritos, como descricao, getCodigo e escreverFicheiro, personalizam as funcionalidades para incluir esses novos atributos. Essa especialização demonstra a flexibilidade da abordagem orientada a objetos, permitindo a criação de classes específicas enquanto reutiliza a estrutura base.

```
class LivroEducativo : public Geral
{
    private:
    string isbn;
    int Nivel_Educacional; // 1 - Fundamental , 2 - Medio, 3 - Superior
    string Materia; // Matemática, história, física, etc..

    public:
        LivroEducativo(string categoria,string titulo, string autor, int ano, bool disponivel,string isbn,int Nivel_Educaci
        virtual ~LivroEducativo();
        void descricao() const override;
        string getCodigo() const override;
        bool escreverFicheiro(ofstream& file) override;
        void AlterarInformacaoDoLivro() override;
```

Figura 9 - Livros educativos

Pessoa.h

A classe Pessoa representa um utilizador da biblioteca, com informações e comportamentos relacionados aos empréstimos, reservas, multas e categoria. Ela é projetada para ser uma classe base para ser estendida por tipos específicos de pessoas, como estudantes, professores, ou outros tipos de leitores.

Atributos

- nome: Armazena o nome da pessoa.
- NIF: Número de Identificação Fiscal, um identificador único.
- totalMultaPorPagar: Total de multa que a pessoa tem por pagar.
- totalMultaPago: Total de multa que já foi paga.
- NumeroDeEmprestimosTotal: Número total de empréstimos feitos pela pessoa.
- NumeroDeEmprestimosAtivos: Quantidade de empréstimos que estão atualmente ativos.
- NumeroDeReservas: Número de reservas feitas pela pessoa.
- categoria: Categoria à qual a pessoa pertence (por exemplo, estudante, professor).
- EmprestimosUser: Lista de empréstimos feitos pela pessoa.
- Reservas: Lista de reservas feitas pela pessoa.

```
class Pessoa
{
    protected:
    string nome;
    string NIF;
    int totalMultaPorPagar;
    int totalMultaPago;
    int NumeroDeEmprestimosTotal;
    int NumeroDeEmprestimosAtivos;
    int NumeroDeReservas;
    string categoria;

    vector<Emprestimo> EmprestimosUser;
    vector<Emprestimo> Reservas;

    public:
        // Construtor e destruidor
        Pessoa(string nome, string NIF, int NumeroDeEmprestimosTotal, int NumeroDeEmprestimosAtivos, int totalMultaPorPagar,
        virtual ~Pessoa();
```

Figura 10 - classe Pessoa

Métodos

1. Construtor:
    - O construtor inicializa os atributos da pessoa, incluindo empréstimos e reservas.
2. Métodos Virtuais e Implementados:
    - descricao(): Método virtual sobrescrito para fornecer uma descrição detalhada do utilizador.
    - escreverFicheiro(ofstream& file): Método virtual que permite gravar as informações da pessoa em um arquivo.
    - Métodos de Acesso (Getters): Métodos como getNome(), getNIF(), getCategoria(), e getNumeroDeEmprestimosTotais() fornecem acesso às informações do utilizador.

3. Métodos Utilitários:
    - decrementarEmprestimosAtivos(): Diminui o número de empréstimos ativos.
    - removerReserva() e adicionarReserva(): Métodos para adicionar e remover reservas feitas pelo utilizador.
    - adicionarReservaPelaLeitura() e

adicionarEmprestimoPelaLeitura(): Métodos para adicionar reservas ou empréstimos, com base em entradas do sistema.

- listarEmprestimos() e listarReservas(): Listam os empréstimos e as reservas realizadas pela pessoa.
- incrementarMulta(): Aumenta o valor da multa devida pela pessoa.

4. Notificações:
    - EnviarNotificacoesdeAtraso(): Método virtual utilizado nas subclasses para notificar o utilizador sobre atrasos.
    - EnviarNotificacoesdeExlusaoDeReserva() e EnviarNotificacoesdeAquisicaoDaReserva(): Envia notificações sobre o estado das reservas.

5. Métodos Puramente Virtuais:
    - getPrazoDevolucao(): Determina o prazo de devolução de um livro, dependendo da categoria do livro.
    - PodeReservar(), PodeEmprestar(): Verificam se a pessoa pode realizar reservas ou empréstimos, isso acontece devido a quantidade de livros reservas e emprestados que pode ter ao mesmo tempo.
    - getLivrosMaximos() e alterarLivrosMaximos(): Determinam o número máximo de livros que a pessoa pode pegar emprestado e alteram esse limite.
    - alterarDescontos() e getDescontoMulta(): Definem e retornam descontos aplicáveis nas multas dos leitores.

    - calcularMultaTotal(): Calcula o valor total da multa devida pela pessoa.

- incrementarEmprestimosAtivos(): Aumenta o número de empréstimos ativos da pessoa.
- addEmprestimo(): Adiciona um empréstimo à lista de empréstimos da pessoa.

Exemplo de subclasse Pessoa - Estudante

A classe Estudante é uma subclasse da classe Pessoa, que define atributos e comportamentos específicos para estudantes. Ela limita o número de livros que o estudante pode emprestar e oferece um determinado desconto sobre as multas. Além disso, a classe sobrescreve vários métodos da classe base para personalizar a lógica de empréstimos, reservas, cálculo de multas e notificação de atrasos. Essa especialização permite que as regras e permissões para estudantes sejam distintas das de outros tipos de utilizadores, mantendo a flexibilidade na gestão da biblioteca.

```
class Estudante : public Pessoa
{
private:
    int livrosMaximos; // 5
    double descontoMulta; // 0.2

public:
    Estudante(string nome, string NIF, int NumeroDeEmprestimosTotal, int NumeroDeEmprestimosAtivos, int totalMultaPorPagar,
    virtual ~Estudante();

    void descricao() const override;
    int getPrazoDevolucao(string categoriaLivro) const override;
    bool PodeReservar() const override;
    bool PodeEmprestar() const override;
    int getLivrosMaximos() override;
    void alterarLivrosMaximos(int max) override;
    void alterarDescontos(int desc) override;
    double getDescontoMulta() override;
    double calcularMultaTotal() override;
    void incrementarEmprestimosAtivos() override;
    void addEmprestimo(Emprestimo& emprestimo) override;
    bool escreverFicheiro(ofstream& file) override;
    void EnviarNotificacoesdeAtraso() override;
```

Figura 11 - Estudante

Emprestimo.h

A classe Emprestimo representa um empréstimo de um livro realizado por um leitor na biblioteca. Ela armazena informações detalhadas sobre o empréstimo, como o leitor, o livro, as datas de empréstimo e devolução, além de permitir operações como comparação entre empréstimos.

Atributos

- NIFLeitor: O Número de Identificação Fiscal do leitor que realizou o empréstimo.
- nomeLeitor: O nome do leitor que fez o empréstimo.
- categoriaLivro: A categoria do livro emprestado (por exemplo, Educativo, Ficcao, Revista etc.)
- categoriaLeitor: A categoria do leitor (por exemplo, estudante, professor, etc.).
- tituloLivro: O título do livro emprestado.
- idLivro: O identificador único do livro.
- dataEmprestimo: A data em que o empréstimo foi realizado, armazenada como um time_t, que representa o tempo em segundos desde 1º de janeiro de 1970.
- dataDevolucao: A data em que o livro deve ser devolvido, também armazenada como um time_t.

```
class Emprestimo {

private:
    string NIFLeitor;     // ID do leitor que realizou o empréstimo
    string nomeLeitor;    // Nome do leitor
    string categoriaLivro;
    string categoriaLeitor;
    string tituloLivro;
    string idLivro;
    time_t dataEmprestimo;
    time_t dataDevolucao;
```

Figura 12 - classe Empréstimos

Métodos

1. Construtor :

    - O construtor inicializa todos os atributos da classe, incluindo as datas do empréstimo e da devolução.

   Métodos de Acesso (Getters):

    - getNifEmprestimo(): Retorna o NIF do leitor que realizou o empréstimo.
    - getIDLivroEmprestimo(): Retorna o ID do livro emprestado.
    - getCategoriaLeitorEmprestimo(): Retorna a categoria do leitor.
    - getCategoriaLivroEmprestimo(): Retorna a categoria do livro.
    - getDataDevolucaoEmprestimo(): Retorna a data de devolução do empréstimo.
2. Métodos Adicionais:

    - Descricao(): Exibe uma descrição detalhada do empréstimo, incluindo o leitor, livro, e datas de empréstimo e devolução.
    - escreverFicheiro(ofstream& file): Grava os detalhes do empréstimono ficheiro empréstimos.txt ou .csv.

3. Operadores:

    - operator== Sobrescreve o operador de comparação para verificar se dois empréstimos são iguais com base em seus atributos. Ao definir esse operador, basicamente está a ser dito ao compilador como comparar dois objetos Empréstimo. Sem ele, o compilador não sabe como comparar dois objetos dessa classe e como resultado o código não compila e termina com erros.

```
bool Emprestimo::operator==(const Emprestimo& other) const {
    return (NIFLeitor == other.NIFLeitor &&
            idLivro == other.idLivro &&
            categoriaLivro == other.categoriaLivro &&
            categoriaLeitor == other.categoriaLeitor &&
            tituloLivro == other.tituloLivro &&
            dataEmprestimo == other.dataEmprestimo &&
            dataDevolucao == other.dataDevolucao);
}
```

Figura 13 - bool operator

Uteis.h

A classe Uteis contém métodos utilitários que facilitam a manipulação de dados em várias partes do sistema de gestão da biblioteca. Esses métodos são usados para operações como criação de livros e utilizadores, manipulação de empréstimos, leitura e gravação de arquivos, e ajustes nos limites de empréstimos e descontos para diferentes categorias de utilizadores e livros.

Atributos:

- map<string, int> limitesPorCategoria: Este atributo armazena os limites de empréstimos por categoria de livro ou utilizador. A chave é uma string (nome da categoria), e o valor é um número inteiro que representa o limite de empréstimos permitido para essa categoria.

- map<string, double> descontoPorCategoria: Armazena o desconto aplicado sobre as multas, com base na categoria do utilizador. A chave é a categoria do utilizador (como "Estudante", "Professor", etc.), e o valor é o valor do desconto (em formato decimal, como 0.2 para 20%).

Construtores e Destruidores:

- Construtor (Uteis()): Inicializa a classe Uteis. Não tem implementação fornecida na declaração, mas configura os mapas vazios.
- Destruidor (virtual ~Uteis()): Libera recursos, embora não haja detalhes fornecidos na declaração.

Métodos:

- CriarLivroUser(Biblioteca& bib): Permite a criação de um livro pelo utilizador. Usa a referência de um objeto Biblioteca para interagir com o sistema de livros.

- LivroInfo(int opcao, string categoria, Biblioteca& biblioteca): Exibe informações detalhadas sobre um livro, com base na opção e na categoria do livro fornecida.

- LerLivrosPorCategoria(Biblioteca& bib, string extensaoArquivo): Lê informações sobre os livros de uma categoria a partir de um arquivo, dependendo da extensão do arquivo especificado (como .txt ou .csv).

- LerPessoasPorCategoria(Biblioteca& bib, string extensaoArquivo): Similar ao método anterior, mas lê dados de pessoas (utilizadores) a partir de um arquivo.

- LerEmprestimos(Biblioteca& biblioteca, string extensaoArquivo): Lê informações sobre empréstimos a partir de um arquivo.

- MudarLimiteLivros(Biblioteca& bib): Permite ao administrador alterar o limite de livros que podem ser emprestados, baseado na categoria de livros ou utilizadores.

- addLimiteLivros(string categoria, int limite): Adiciona um limite de empréstimos para uma categoria específica.

- LerLimitesPorCategoria(string extensaoArquivo): Lê os limites de empréstimos para cada categoria a partir de um arquivo.

- gravarLimitesPorCategoria(string extensaoArquivo): Grava os limites de empréstimos por categoria em um arquivo.

- alterarLivroInfo(Biblioteca& bib): Permite a alteração das informações de um livro na biblioteca.

- MudarDesconto(Biblioteca& bib): Permite alterar o valor do desconto de multa para as diferentes categorias de utilizadores.

- addDesconto(string categoria, double desconto): Adiciona um desconto para uma categoria de utilizador.

- LerDesconto(string extensaoArquivo): Lê o desconto por categoria a partir de um arquivo.

- gravarDesconto(string extensaoArquivo): Grava os descontos em um arquivo.

- CriarUser(Biblioteca& bib): Permite a criação de um novo utilizador (leitor) na biblioteca.

- UserInfo(int opcao, string categoria, Biblioteca& biblioteca): Exibe informações detalhadas sobre um utilizador, com base na categoria e na

opção escolhida.

- EmprestimoFuncaoPrincipal(Biblioteca& bib): Executa a principal função de empréstimo, é a função onde é chamada as outras funções necessárias para se realizar o empréstimo.

- ListarPorCategoriaLivro(Biblioteca& bib, bool search): Lista os livros de uma categoria específica na biblioteca. A variável search pode ser usada para indicar se é uma busca filtrada.

- ListarPorCategoriaUtilizador(Biblioteca& bib, bool search): Lista os utilizadores (leitores) de uma categoria, com a possibilidade de realizar uma busca filtrada.

- ConsultarHistoricoUtilizador(Biblioteca& bib): Permite consultar o histórico de empréstimos e reservas de um utilizador.

- RelatorioTipoDeLivro(Biblioteca& bib): Gera um relatório detalhado sobre os tipos de livros da biblioteca.

- ConsultarHistoricoDeReservas(Biblioteca& bib): Permite visualizar o histórico de reservas feitas por utilizadores.

- DevolverLivro(Biblioteca& bib): Gere a devolução de um livro emprestado.

- GravarGeral(Biblioteca& bib): Realiza a gravação de dados gerais da biblioteca (livros, empréstimos, etc.) em arquivos.

- LerGeral(Biblioteca& bib): Lê dados gerais da biblioteca a partir de

arquivos.

Funções auxiliares:

- retorno RetornarType_String() e retorno RetornarType_String_User(): Essas funções retornam uma estrutura retorno que contém um tipo e uma categoria, que podem ser usadas para capturar o tipo de dados (como livro ou utilizador) e a categoria associada.

Conclusão

Este projeto teve como finalidade consolidar habilidades e técnicas na linguagem C++.

Durante o desenvolvimento, destacaram-se as técnicas de herança e polimorfismo, que permitiram estruturar o código de forma modular, reutilizável e extensível. A classe base Geral, por exemplo, foi projetada para representar atributos e comportamentos comuns a todos os tipos de livros, enquanto suas subclasses, como LivroEducativo, adicionaram características específicas, como o nível educacional e a matéria abordada. Essa abordagem enfatizou a reutilização de código e a segregação lógica de responsabilidades, simplificando a manutenção e a adição de novos tipos de livros no sistema.

O polimorfismo foi empregue para permitir que métodos, como descricao e AlterarInformacaoDoLivro, fossem definidos na classe base como virtuais puras e implementados de maneira específica em cada subclasse. Esse recurso proporcionou flexibilidade e garantiu que o comportamento adequado fosse executado em tempo de execução, dependendo do tipo real do objeto.

Além disso, a classe Pessoa exemplificou a aplicação de herança para diferenciar perfis de usuários da biblioteca, como Estudante, com atributos e métodos que atendem às suas necessidades específicas, como limites de empréstimos e descontos em multas. A implementação de métodos virtuais puros assegurou que cada tipo de usuário pudesse definir seu próprio comportamento para funcionalidades críticas, como a gestão de empréstimos e reservas.

No geral, o projeto demonstrou a eficiência do paradigma orientado a objetos na solução de problemas complexos, como o gerenciamento de

bibliotecas. O uso de herança, polimorfismo e outros recursos avançados da linguagem C++ não apenas enriqueceu a estrutura do programa, mas também reforçou a importância desses conceitos na engenharia de software. O sistema desenvolvido é escalável, organizado e preparado para futuras expansões, consolidando as competências adquiridas ao longo do curso e sua aplicação prática em projetos reais. Embora tenhamos enfrentado alguns obstáculos e problemas com erros de programação, sempre nos esforçámos para superá-los. Como resultado, este projeto contribuiu significativamente para melhorar o nosso desempenho na programação.