Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Departamento de Informática

Unidade Curricular: Programação Orientada a Objetos

Relatório do Projeto:

Sistema de Gestão de Bibliotecas

Trabalho prático de Programação Orientada a Objetos

(2024/2025)

Realizado por: Guilherme Félix pv25172

Viseu, 2025

Índice

Índice de imagens

1. Introdução

Para a realização deste Projeto Prático da cadeira de Programação Orientada a Objetos, foi-nos proposto a criação de um Sistema de Gestão de Bibliotecas, aplicando conceitos dados em aula como a herança e polimorfismo para modelar diferentes tipos de livros. Neste projeto usamos diferentes tipos de livro como a ficção, científicos, educativos, revistas, jornais e também diferentes tipos de leitores como os estudantes, professores, seniores, etc. Na realização do projeto, fizemos um sistema que vai permitir gerir empréstimos, reservas, multas e relatórios numa biblioteca, onde irão acontecer alterações dinâmicas referentes ao tipo de utilizador e livro. O objetivo deste trabalho é melhorar os nossos conhecimentos na cadeira de POO, desenvolvendo uma solução eficiente e fácil de manutenção de uma biblioteca.

2. Desenvolvimento

O sistema de gestão da biblioteca foi desenvolvido para permitir a administração eficiente de livros, leitores e empréstimos.

2.1 Funcionalidades Principais

2.1.1 Gestão de Leitores (Pessoas)

- Adicionar novos leitores à biblioteca, classificados como:
  - Comum
  - Estudante
  - Professor
  - Sénior
- Editar informações de um leitor existente.
- Listar leitores filtrados por categoria.
- Aplicação de regras específicas, como descontos em multas e limite de reservas por tipo de leitor.

```
class Pessoa {
    protected:
        int nif;
        string nome;
        string tipo;
        int NEmprestimosTotal;
        int NEmprestimosAtivos;
        int NReservas;
        double MultaPorPagar;
        double MultaPaga;
        vector<Emprestimo> historicoEmprestimos;

public:
    Pessoa(int nif, const string& nome, const string& tipo, int NEmprestimosTotal,
           int NEmprestimosAtivos, int NReservas, double MultaPorPagar, double MultaPaga);
    virtual ~Pessoa();
    virtual void mostrarInformacoes() const = 0;
    virtual double calcularMulta(int diasAtraso) const = 0;
    virtual int getMaxReservas() const = 0;
    virtual void notificarAtraso(const string& tituloLivro, int diasAtraso) const = 0;

    int getNIF() const;
    string getTipo() const;
    string getNome() const { return nome; }
    double getMultaPorPagar() const { return MultaPorPagar; }

    void setNome(const string& novoNome) { nome = novoNome; }

    void adicionarMulta(double valor);
    void decrementarReservas();

    void adicionarEmprestimoAoHistorico(const Emprestimo& emprestimo);
    void mostrarHistoricoEmprestimos() const;
```

Figura 1 - Estrutura Pessoa

2.1.2 Gestão de Livros

```
class Livro {
    protected:
        string tipo; // Tipo do livro (Ficcao, Cientifico, Educativo, Revista, Jornal)
        string titulo;
        string autor;
        int anoPublicacao;
        bool disponivel;

    public:
        Livro(const string& tipo, const string& titulo, const string& autor, int anoPublicacao, bool disponivel);
        virtual ~Livro();

        virtual void mostrarInformacoesLivro() const;
        string getTipo() const { return tipo; }
        string getTitulo() const { return titulo; }
        string string Livro::getTitulo() autor; }
        int getAnoPublicacao() const { return anoPublicacao; }
        bool estaDisponivel() const { return disponivel; }

        void setTitulo(const string& novoTitulo) { titulo = novoTitulo; }
        void setAutor(const string& novoAutor) { autor = novoAutor; }
        void setAnoPublicacao(int novoAno) { anoPublicacao = novoAno; }
        void setDisponibilidade(bool status) { disponivel = status; }
};
```

Figura 2 - Classe Livro

- Adicionar livros à biblioteca, classificando-os em:
    - Ficção
    - Científico
    - Educativo
    - Revista
    - Jornal
- Editar informações de um livro existente.
- Listar livros filtrados por categoria.
- Verificar a disponibilidade de um livro.

2.1.3 Gestão de Empréstimos

```
class Emprestimo {
private:
    Pessoa* leitor;
    Livro* livro;
    string dataEmprestimo;
    string dataDevolucao;

public:
    // Construtor da classe Emprestimo
    Emprestimo(Pessoa* leitor, Livro* livro, const string& dataEmprestimo);

    // Método para manipulação de empréstimos
    Pessoa* getLeitor() const;
    Livro* getLivro() const;
    string getDataEmprestimo() const;
    string getDataDevolucao() const;

    void setDataDevolucao(const string& data);
    bool estaAtrasado() const;

    static string calcularDataDevolucao(const string& dataEmprestimo);


    void exibirInfo() const;
};
```

Figura 3 - Classe Emprestimo

```
========================================
        GESTÃO DE EMPRÉSTIMOS
========================================
1. Realizar Empréstimo
2. Reservar Livro
3. Devolver Livro
4. Listar Empréstimos
5. Prorrogar Empréstimo
6. Histórico de Empréstimos
0. Voltar ao Menu Principal
========================================
Escolha uma opção:
```

Figura 4 - Menu Gestão Empréstimos

- Realizar empréstimos de livros para leitores.
- Controlar o prazo de devolução de um livro.
- Prorrogação de empréstimos permitida apenas para Estudantes e Professores.
- Aplicação de multas em caso de atraso na devolução de um livro.

2.1.4 Sistema de Reservas

- Leitores podem reservar um livro caso ele já esteja emprestado.

- Fila de reservas implementada com *queue* para garantir a ordem de espera.
- Estudantes possuem prioridade na reserva de livros educativos.
- Reservas automáticas: Ao devolver um livro reservado, ele é imediatamente emprestado ao próximo leitor na fila.

2.1.5 Relatórios Automáticos

- **Relatório de Livros por Categoria**:
  - Lista todos os livros organizados pela categoria específica.
- **Relatório de Empréstimos**:
  - Classifica os empréstimos por tipo de livro e identifica o tipo de leitor que mais requisita cada categoria.
- **Relatório de Multas**:
  - Lista os leitores que receberam multas e o valor acumulado.

2.1.6 Gravação e Recuperação de Dados

- Leitura automática dos dados de leitores e Livros a partir de ficheiros .txt no início da execução.
- Exportação de dados completos (Leitores, Livros, Empréstimos e Relatórios) para um ficheiro escolhido pelo utilizador.
- Relatórios gerados são incluídos no ficheiro de exportação para facilitar a análise.

2.1.7 Interface do Utilizador

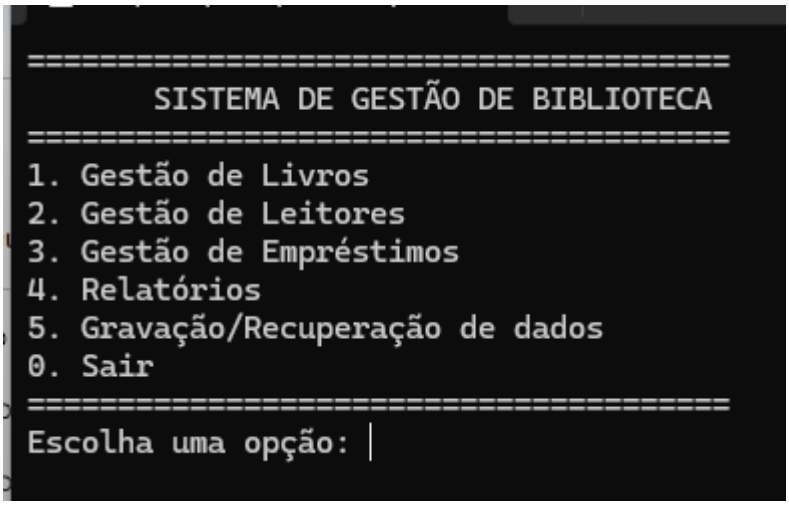

Figura 5 - Menu Principal

- Sistema baseado em **menus interativos** para facilitar a navegação.
- Opções numeradas para cada funcionalidade.
- Entrada validada para evitar erros do utilizador.
- Limpeza da tela (system("cls")) para uma melhor experiência visual.

2.2 Biblioteca

A classe Biblioteca é o núcleo do sistema, sendo responsável pela gestão centralizada dos livros, leitores, empréstimos, reservas, relatórios e gravação de dados.

2.2.1 Carregar pessoas e livros dos ficheiros Pessoas.txt e Livros.txt

```
// Carregar pessoas e livros dos ficheiros Pessoas.txt e Livros.txt
void carregarPessoas(const string& ficheiro); //Feito
void carregarLivros(const string& ficheiro); //Feito
```

Figura 6 - Funções Pessoas e Livros.txt

- void carregarPessoas(const string& ficheiro);
- void carregarLivros(const string& ficheiro);

2.2.2 Livros

```
// Métodos para gestão de livros
void adicionarLivro(Livro* livro); //Feito
void editarLivro(Livro* livro); //Feito
void listarLivrosPorTipo(const string& tipo) const; //Feito
Livro* procurarLivro(const string& titulo) const; //Feito
```

Figura 7 - Funções Livro

- void adicionarLivro(Livro* livro);
- void editarLivro(Livro* livro);
- void listarLivrosPorTipo(const string& tipo) const;
- Livro* procurarLivro(const string& titulo) const;

2.2.3 Leitores (Pessoas)

```
// Métodos para gestão de leitores
void adicionarLeitor(Pessoa* leitor); //Feito
void editarLeitor(Pessoa* leitor); //Feito
void listarLeitoresPorTipo(const string& tipo) const; //Feito
Pessoa* procurarLeitor(int nif) const; //Feito
```

Figura 8 - Funções Leitor

- void adicionarLeitor(Pessoa* leitor);
- void editarLeitor(Pessoa* leitor) ;
- void listarLeitoresPorTipo(const string& tipo) const;
- Pessoa* procurarLeitor(int nif) const;

2.2.4 Empréstimos

-

```
// Métodos para gestão de empréstimos
void realizarEmprestimo(Pessoa* leitor, Livro* livro, const string& dataEmprestimo); //Feit
void reservarLivro(Pessoa* leitor, const string& tituloLivro); //Feito
void devolverLivro(int nifLeitor, const string& tituloLivro, const string& dataDevolucao);
void listarEmprestimos() const; //Feito
void prorrogarEmprestimo(int nifLeitor, const string& tituloLivro, const string& novaData);
```

Figura 9 - Funções Empréstimos

- void realizarEmprestimo(Pessoa* leitor, Livro* livro, const string& dataEmprestimo); (Fig.)
- void reservarLivro(Pessoa* leitor, const string& tituloLivro); (Fig.)
- void devolverLivro(int nifLeitor, const string& tituloLivro, const string& dataDevolucao);
- void listarEmprestimos() const;
- void prorrogarEmprestimo(int nifLeitor, const string& tituloLivro, const string& novaData);

2.2.5 Relatórios

```
// Métodos para gestão de relatórios
void gerarRelatorioEmprestimos(ofstream* arquivo = nullptr) const;
void gerarRelatorioMultas(ofstream* arquivo = nullptr) const;
void gerarRelatorioLivrosCategoria(ofstream* arquivo = nullptr) const;
```

Figura 10 - Funções Relatórios

- void gerarRelatorioEmprestimos(ofstream* arquivo = nullptr) const;
- void gerarRelatorioMultas(ofstream* arquivo = nullptr) const;

- void gerarRelatorioLivrosCategoria(ofstream* arquivo = nullptr) const;

2.2.6 Persistência de Dados

```
// Persistência de dados
void gravarDados(const string& ficheiro) const;
void gravarDadosEmFicheiro();
```

Figura 11 - Funções de gravar dados

- void gravarDados(const string& ficheiro) const;
- void gravarDadosEmFicheiro();

2.3 Pessoa

```
// Construtor da classe base Pessoa
Pessoa::Pessoa(int nif,
               const string& nome,
               const string& tipo,
               int NEmprestimosTotal,
               int NEmprestimosAtivos,
               int NReservas,
               double MultaPorPagar,
               double MultaPaga)
    : nif(nif), nome(nome), tipo(tipo),
    NEmprestimosTotal(NEmprestimosTotal),
    NEmprestimosAtivos(NEmprestimosAtivos),
    NReservas(NReservas), MultaPorPagar(MultaPorPagar),
    MultaPaga(MultaPaga) {}

Pessoa::~Pessoa() {}
```

Figura 12 - Construtor Pessoa

Define a hierarquia dos usuários do sistema, aplicando polimorfismo para diferenciar regras para leitores comuns, estudantes, professores e seniores.

2.3.1 Atributos

NIF- Identificação única do leitor.
Nome - Nome do leitor.

Tipo - Tipo de leitor (Comum, Estudante, Professor, Senior).
NEmprestimosTotal - Número total de empréstimos já realizados.
NEmprestimosAtivos - Número de empréstimos ativos no momento.
NReservas - Quantidade de reservas ativas.
MultaPorPagar - Valor da multa pendente do leitor.
MultaPaga - Valor total já pago em multas.
historicoEmprestimos - Armazena o histórico dos empréstimos realizados.

2.3.2 Métodos principais

calcularMulta() - Método virtual puro que aplica taxas de multa personalizadas para cada tipo de leitor.
getMaxReservas() - Define o número máximo de reservas permitidas, dependendo da subclasse.
notificarAtraso() - Método virtual puro para notificar o leitor sobre um atraso.
mostrarInformacoes() - Exibe os dados do leitor.
adicionarMulta(double valor) - Adiciona um valor de multa ao leitor.
adicionarEmprestimoAoHistorico() - Armazena um empréstimo no histórico do leitor.

2.3.3 Subclasse Leitor Comum

```
//-------------------- Subclasse: Comum --------------------//
class Comum : public Pessoa {
    private:
        int maxLivros;
        double descontoMulta;

    public:
        Comum(int nif, const string& nome, int NEmprestimosTotal, int NEmprestimosAtivos,
              int NReservas, double MultaPorPagar, double MultaPaga);
        void mostrarInformacoes() const override;
        double calcularMulta(int diasAtraso) const override;
        int getMaxReservas() const override { return 2; }
        void notificarAtraso(const string& tituloLivro, int diasAtraso) const override;
        friend class Biblioteca;
};
```

Figura 13 - Subclasse Leitor Comum

- Leitores sem vínculos específico.
- Podem reservar até 2 livros.
- Multas aplicadas sem desconto.

2.3.4 Subclasse Estudante

```
//-------------------- Subclasse: Estudante --------------------//
class Estudante : public Pessoa {
private:
    int maxLivros;
    double descontoMulta;
    string curso;

public:
    Estudante(int nif, const string& nome, int NEmprestimosTotal, int NEmprestimosAtivos,
              int NReservas, double MultaPorPagar, double MultaPaga, const string& curso);
    void mostrarInformacoes() const override;
    double calcularMulta(int diasAtraso) const override;
    string getCurso() const { return curso; }
    void setCurso(const string& novoCurso) { curso = novoCurso; }
    int getMaxReservas() const override { return 5; }
    void notificarAtraso(const string& tituloLivro, int diasAtraso) const override;
    friend class Biblioteca;
};
```

Figura 14 - Subclasse Pessoa Estudante

- Leitores que são estudantes matriculados.
- Podem reservar até 5 livros.
- Recebem um desconto de 20% nas multas.
- Podem prorrogar empréstimos.
- Possuem um campo adicional curso.

2.3.5 Subclasse Senior

```
//-------------------- Subclasse: Senior --------------------//
class Senior : public Pessoa {
    private:
        int maxLivros;
        double descontoMulta;
        int idade;

    public:
        Senior(int nif, const string& nome, int NEmprestimosTotal, int NEmprestimosAtivos,
               int NReservas, double MultaPorPagar, double MultaPaga, int idade);
        void mostrarInformacoes() const override;
        double calcularMulta(int diasAtraso) const override;
        int getIdade() const { return idade; }
        void setIdade(int novaIdade) { idade = novaIdade; }
        int getMaxReservas() const override { return 3; }
        void notificarAtraso(const string& tituloLivro, int diasAtraso) const override;
        friend class Biblioteca;
};
```

Figura 15 - Subclasse Pessoa Senior

- Leitores maiores de idade com regras especiais.
- Podem reservar até 3 livros.
- Recebem um desconto de 30% nas multas.
- Possuem um campo adicional idade.

2.3.6 Subclasse Professor

```
//-------------------- Subclasse: Professor --------------------//
class Professor : public Pessoa {
    private:
        int maxLivros;
        double descontoMulta;
        string departamento;

    public:
        Professor(int nif, const string& nome, int NEmprestimosTotal, int NEmprestimosAtivos,
                  int NReservas, double MultaPorPagar, double MultaPaga, const string& departamento);
        void mostrarInformacoes() const override;
        double calcularMulta(int diasAtraso) const override;
        string getDepartamento() const { return departamento; }
        void setDepartamento(const string& novoDepartamento) { departamento = novoDepartamento; }
        int getMaxReservas() const override { return 4; }
        void notificarAtraso(const string& tituloLivro, int diasAtraso) const override;
        friend class Biblioteca;
};
```

Figura 16 - Subclasse Pessoa Professor

- Leitores que são professores afiliados à instituição.
- Podem reservar até 4 livros.
- Recebem um desconto de 50% nas multas.
- Possuem um campo adicional departamento.
- Podem prorrogar empréstimos.

2.4 Livro

```
class Livro {
    protected:
        string tipo; // Tipo do livro (Ficcao, Cientifico, Educativo, Revista, Jornal)
        string titulo;
        string autor;
        int anoPublicacao;
        bool disponivel;

    public:
        Livro(const string& tipo, const string& titulo, const string& autor, int anoPublicacao, bool disponivel);
        virtual ~Livro();

        virtual void mostrarInformacoesLivro() const;
        string getTipo() const { return tipo; }
        string getTitulo() const { return titulo; }
        string getAutor() const { return autor; }
        int getAnoPublicacao() const { return anoPublicacao; }
        bool estaDisponivel() const { return disponivel; }

        void setTitulo(const string& novoTitulo) { titulo = novoTitulo; }
        void setAutor(const string& novoAutor) { autor = novoAutor; }
        void setAnoPublicacao(int novoAno) { anoPublicacao = novoAno; }
        void setDisponibilidade(bool status) { disponivel = status; }
};
```

Figura 17 - Classe Livro

O arquivo dos Livro define a classe base Livro, que representa os livros disponíveis na biblioteca, e as suas subclasses específicas (LivroFiccao, LivroCientifico, LivroEducativo, Revista e Jornal).

2.4.1 Atributos

titulo - Nome do livro.
autor - Nome do autor.
anoPublicacao - Ano em que o livro foi publicado.
disponivel - Indica se o livro pode ser emprestado.
tipo - Define a categoria do livro.

2.4.2 Métodos principais

getTitulo() - Retorna o título do livro.
getAutor() - Retorna o autor do livro.
getAnoPublicacao() - Retorna o ano de publicação.
estaDisponivel() - Verifica se o livro pode ser emprestado.
setDisponibilidade(bool estado) - Define se o livro está disponível ou emprestado.
mostrarInformacoesLivro() - Exibe detalhes do livro.

2.4.3 Subclasses

```
//-------------------- Subclasse: LivroFiccao --------------------//
class LivroFiccao : public Livro {
    private:
        string genero;
        string isbn;

    public:
        LivroFiccao(const string& titulo, const string& autor, int anoPublicacao, bool disponivel, const string& genero, const string& isbn);
        void mostrarInformacoesLivro() const override ;
        string getGenero() const { return genero; }
        void setGenero(const string& novoGenero) { genero = novoGenero; }
        friend class Biblioteca;
};

//-------------------- Subclasse: LivroCientifico --------------------//
class LivroCientifico : public Livro {
    private:
        string areaCientifica;
        string isbn;

    public:
        LivroCientifico(const string& titulo, const string& autor, int anoPublicacao, bool disponivel, const string& areaCientifica, const string& isbn);
        void mostrarInformacoesLivro() const override;
        string getAreaCientifica() const { return areaCientifica; }
        void setAreaCientifica(const string& novaArea) { areaCientifica = novaArea; }
        friend class Biblioteca;
```

Figura 18 - Subclasse Livro Ficção e Subclasse Livro Científico

```
//-------------------- Subclasse: LivroEducativo --------------------//
class LivroEducativo : public Livro {
    private:
        string nivelEducacional;
        string isbn;

    public:
        LivroEducativo(const string& titulo, const string& autor, int anoPublicacao, bool disponivel, const string& nivelEducacional, const string& isbn);
        void mostrarInformacoesLivro() const override;
        string getNivelEducacional() const { return nivelEducacional; }
        void setNivelEducacional(const string& novoNivel) { nivelEducacional = novoNivel; }
        friend class Biblioteca;
};

//-------------------- Subclasse: Revista --------------------//
class Revista : public Livro {
    private:
        int numeroEdicao;

    public:
        Revista(const string& titulo, const string& autor, int anoPublicacao, bool disponivel, int numeroEdicao);
        void mostrarInformacoesLivro() const override;
        int getNumeroEdicao() const { return numeroEdicao; }
        void setNumeroEdicao(int novoNumero) { numeroEdicao = novoNumero; }
        friend class Biblioteca;
};

//-------------------- Subclasse: Jornal --------------------//
class Jornal : public Livro {
    private:
        string dataPublicacao;

    public:
        Jornal(const string& titulo, const string& autor, int anoPublicacao, bool disponivel, const string& dataPublicacao);
        void mostrarInformacoesLivro() const override;
        string getDataPublicacao() const { return dataPublicacao; }
        void setDataPublicacao(const string& novaData) { dataPublicacao = novaData; }
        friend class Biblioteca;
};
```

Figura 19 - Subclasse Livro Educativo/ Subclasse Livro Revista e Subclasse Livro Jornal

- **LivroFiccao** - Possui um campo adicional genero, isbn.
- **LivroCientifico** - Possui um campo adicional areaCientifica, isbn.
- **LivroEducativo** - Possui um campo adicional nivelEducacional, isbn.
- **Revista** - Possui um campo adicional numeroEdicao.
- **Jornal** - Possui um campo adicional dataPublicacao.

2.5 Empréstimo

```
class Emprestimo {
private:
    Pessoa* leitor;
    Livro* livro;
    string dataEmprestimo;
    string dataDevolucao;

public:
    // Construtor da classe Emprestimo
    Emprestimo(Pessoa* leitor, Livro* livro, const string& dataEmprestimo);

    // Método para manipulação de empréstimos
    Pessoa* getLeitor() const;
    Livro* getLivro() const;
    string getDataEmprestimo() const;
    string getDataDevolucao() const;

    void setDataDevolucao(const string& data);
    bool estaAtrasado() const;

    static string calcularDataDevolucao(const string& dataEmprestimo);


    void exibirInfo() const;
};
```

Figura 20 - Classe Emprestimo

O módulo Empréstimo gere os empréstimos de livros, vinculando um livro a um leitor.

2.5.1 Atributos

leitor - Pessoa que pegou o livro emprestado.
livro - Livro emprestado.
dataEmprestimo - Data em que o livro foi retirado.
dataDevolucao - Data prevista para devolução.

2.5.2 Métodos principais

getLeitor() - Retorna o leitor que pegou o livro emprestado.
getLivro() - Retorna o livro emprestado.
getDataEmprestimo() - Retorna a data do empréstimo.
getDataDevolucao() - Retorna a data de devolução.

setDataDevolucao(novaData) - Atualiza a data de devolução (usado na prorrogação de empréstimos).

2.6 Uteis

Este módulo contém funções auxiliares e menus que organizam a interface do sistema. Mantém a execução do programa organizada, garantindo que os dados sejam carregados corretamente antes do usuário interagir com o sistema.

```
class Uteis {
public:
    // Função para mostrar o menu principal
    static void menuPrincipal(Biblioteca& biblioteca);
    static int calcularDiferencaDias(const string& data1, const string& data2);
    static void menuExibirHistoricoEmprestimos(Biblioteca& biblioteca);

private:
    //--------------- Livros ---------------//
    static void menuGestaoLivros(Biblioteca& biblioteca);
    static void menuAdicionarLivro(Biblioteca& biblioteca);
    static void menuEditarLivro(Biblioteca& biblioteca);
    static void menuListarLivros(Biblioteca& biblioteca);
    //--------------- Leitores ---------------//
    static void menuGestaoLeitores(Biblioteca& biblioteca);
    static void menuAdicionarLeitor(Biblioteca& biblioteca);
    static void menuEditarLeitor(Biblioteca& biblioteca);
    static void menuProcurarLeitor(Biblioteca& biblioteca);
    static void menuListarLeitores(Biblioteca& biblioteca);
    static void menuProcurarLivro(Biblioteca& biblioteca);
    //--------------- Emprestimos ---------------//
    static void menuGestaoEmprestimos(Biblioteca& biblioteca);
    static void menuRealizarEmprestimo(Biblioteca& biblioteca);
    static void menuReservarLivro(Biblioteca& biblioteca);
    static void menuDevolverLivro(Biblioteca& biblioteca);
    static void menuListarEmprestimos(Biblioteca& biblioteca);
    static void menuProrrogarEmprestimo(Biblioteca& biblioteca);
    //--------------- Relatórios ---------------//
    static void menuRelatorios(Biblioteca& biblioteca);
    void menuGerarRelatorioMultasPendentes(Biblioteca& biblioteca);

    // Função auxiliar para exibir opções e obter escolha do utilizador
    static int obterEscolha(int min, int max);
};
```

Figura 21 - Classe Uteis

2.6.1 Funções de Interface e Menu

menuPrincipal(Biblioteca&) - Controla a navegação principal do sistema.
menuGestaoLivros(Biblioteca&) - Exibe o menu de gerenciamento de livros.
menuGestaoLeitores(Biblioteca&) - Exibe o menu de gerenciamento de leitores.

menuGestaoEmprestimos(Biblioteca&) - Exibe o menu de empréstimos e reservas.
menuRelatorios(Biblioteca&) - Gera os relatórios de uso da biblioteca.

2.6.2 Funções Auxiliares

limparTela() - Limpa a tela para melhor legibilidade e para ter uma interface limpa.
obterEscolha(int min, int max) - Obtém uma escolha válida dentro de um intervalo.

2.6.3 Gerir Arquivos

carregarPessoas(string ficheiro) - Lê e carrega leitores a partir de um ficheiro Pessoas.txt.
carregarLivros(string ficheiro) - Lê e carrega livros a partir de um ficheiro Livros.txt.

2.7 Main

```
#include "Biblioteca.h"
#include "Uteis.h"
#include <locale.h>
#include <time.h>

int main() {

    srand(0);

    setlocale(LC_ALL, "Portuguese");

    Biblioteca biblioteca;

    // Passa as pessoas do ficheiro para a memória
    biblioteca.carregarPessoas("Pessoas.txt");
    biblioteca.carregarLivros("Livros.txt");
    // Mostra o menu principal
    Uteis::menuPrincipal(biblioteca);

    return 0;
}
```

Figura 22 - Main

O **main.cpp** é o ponto principal do programa. Ele cria a Biblioteca, carrega os dados e exibe o menu principal.

2.7.1 Principais operações

Configuração Inicial

- srand(0); → Inicializa a geração de números aleatórios (se necessário).
- setlocale(LC_ALL, "Portuguese"); → Garante que caracteres especiais sejam mostrados corretamente.

Criação da Biblioteca

- Biblioteca biblioteca;

Carregamento de Dados

- biblioteca.carregarPessoas("Pessoas.txt");
- biblioteca.carregarLivros("Livros.txt");

Exibição do Menu Principal

- Uteis::menuPrincipal(biblioteca);

2.8 Implementação de Reservas e Prioridades

- Fila de reservas (std::queue)
    - Estudantes têm prioridade em Livros Educativos.
    - Gestão de reservas automáticas ao devolver um livro.

2.9 Estruturas de Dados Utilizadas

O projeto utiliza diferentes **estruturas de dados** para garantir eficiência na manipulação das informações.

```
class Biblioteca {
    private:
        vector<Livro*> livros;
        map<int, Pessoa*> leitores;
        vector<Emprestimo> emprestimos;
        map<string, queue<Pessoa*>> reservasNormais;
        map<string, queue<Pessoa*>> reservasPrioritarias;
        map<int, double> multasRegistradas;
```

Figura 23 - Tipos de estruturas de dados utilizados

vector - Utilizado para armazenar listas dinâmicas de livros e empréstimos.
map - Usado para indexação eficiente de leitores por NIF e categorização de relatórios.
queue - Implementação de filas de reservas, para garantir o requisito sobre a ordem de prioridade de certos leitores.

3. Conclusão

Este projeto teve como finalidade consolidar habilidades e técnicas na linguagem C++.
A realização deste Projeto permitiu a aplicação prática dos conceitos fundamentais da cadeira de Programação Orientada a Objetos, como por exemplo a herança e o polimorfismo, que resultou na consolidação dos nossos conhecimentos teóricos/práticos adquiridos nas aulas.
Este trabalho não foi fácil de fazer, mas cheguei ao fim da sua realização e notei melhorias enormes, principalmente na capacidade de resolver problemas mais complexos, preparando-nos assim para desafios futuros no desenvolvimento de software orientado a objetos.