Documentacao - Projeto Final

AUTHOR
Versão 1.0

Visão Geral do Problema

O sistema de locadora é um programa que simula o gerenciamento de filmes e clientes em uma locadora. Ele oferece funcionalidades como cadastro de filmes (fita e DVD), cadastro de clientes, aluguel e devolução de filmes, cálculo do valor total a pagar e geração de relatórios de aluguéis. O objetivo é proporcionar uma experiência intuitiva e eficiente para os usuários, facilitando as operações relacionadas ao universo de locação de filmes.

Solução Proposta

A solução proposta consiste em um conjunto de classes em C++ que modelam os principais elementos do problema: **Locadora**, **Cliente**, **Filme**, **Fita**, **DVD**, etc. O sistema é orientado a objetos e utiliza estruturas de dados como vetores e mapas para armazenar informações relevantes, como o estoque de filmes, clientes cadastrados e locações em curso.

As principais funcionalidades incluem:

- Cadastro e remoção de filmes (fita e DVD) e clientes.
- Aluguel e devolução de filmes.
- Cálculo do valor total a pagar considerando dias de locação.
- Geração de relatórios de alugueis.

O programa também permite a leitura do cadastro inicial de filmes a partir de um arquivo, proporcionando flexibilidade para a gestão do estoque.

Solução Proposta Pricipais Dificuldades

As principais dificuldades enfrentadas incluíram debugging, conexão eficiente entre as classes e lidar com a complexidade inerente do problema. Superar esses desafios aprimorou minhas habilidades de depuração e fortaleceu minha compreensão de hierarquia de classes. Apesar dos obstáculos, o projeto proporcionou aprendizado valioso, refinando minha capacidade de resolver problemas e gerenciar complexidades em projetos de software.

Índice Hierárquico

Hierarquia de Classes

Esta lista de hierarquias está parcialmente ordenada (ordem alfabética):

- Cliente
- Filme
- DVD
- Fita
- Locadora

Índice dos Componentes

Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

- **Cliente (Classe que representa um cliente da locadora)**
- **DVD (Classe que representa um DVD na locadora)**
- **Filme (Classe base abstrata que representa um filme na locadora)**
- **Fita (Classe derivada que representa uma fita de vídeo na locadora)**
- **Locadora (Classe que representa uma locadora de filmes)**

Índice dos Arquivos

Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

- **pds2/projeto final/include/Cliente.hpp**
- **pds2/projeto final/include/DVD.hpp**
- **pds2/projeto final/include/Filme.hpp**
- **pds2/projeto final/include/Fita.hpp**
- **pds2/projeto final/include/Locadora.hpp**
- **pds2/projeto final/src/main.cpp**
- **pds2/projeto final/src/mainTests.cpp**
- **pds2/projeto final/src/modulos/Cliente.cpp**
- **pds2/projeto final/src/modulos/DVD.cpp**
- **pds2/projeto final/src/modulos/Filme.cpp (Arquivo de implementação para a classe Filme)**
- **pds2/projeto final/src/modulos/Fita.cpp (Implementação da classe Fita, derivada da classe Filme)**
- **pds2/projeto final/src/modulos/Locadora.cpp (Implementação da classe Locadora)**
- **pds2/projeto final/tests/testeCliente.cpp**
- **pds2/projeto final/tests/testeDVD.cpp**
- **pds2/projeto final/tests/testeFilme.cpp**
- **pds2/projeto final/tests/testeFita.cpp**

Classes

Referência da Classe Cliente

Classe que representa um cliente da locadora.
`#include <Cliente.hpp>`

Membros Públicos

Cliente (std::string cpf, std::string nome)
*Construtor da classe **Cliente**.*

std::string **getCPF** ()
Obtém o CPF do cliente.

std::string **getNome** ()
Obtém o nome do cliente.

Cliente (std::string cpf, std::string nome)
*Construtor da classe **Cliente**.*

std::string **getCPF** ()
Obtém o CPF do cliente.

std::string **getNome** ()
Obtém o nome do cliente.

Descrição detalhada

Classe que representa um cliente da locadora.

Construtores e Destrutores

Cliente::Cliente (std::string *cpf*, std::string *nome*)

Construtor da classe **Cliente**.

Parâmetros

cpf	CPF do cliente.
nome	Nome do cliente.

Cliente::Cliente (std::string *cpf*, std::string *nome*)

Construtor da classe **Cliente**.

Parâmetros

cpf	CPF do cliente.
nome	Nome do cliente.

Documentação das funções

std::string Cliente::getCPF ()

Obtém o CPF do cliente.

Retorna

String representando o CPF do cliente.

std::string Cliente::getCPF ()

Obtém o CPF do cliente.

Retorna

O CPF do cliente.

std::string Cliente::getNome ()

Obtém o nome do cliente.

Retorna

String representando o nome do cliente.

std::string Cliente::getNome ()

Obtém o nome do cliente.

Retorna

O nome do cliente.

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

pds2/projeto final/include/**Cliente.hpp**
pds2/projeto final/src/modulos/**Cliente.cpp**

Referência da Classe DVD

Classe que representa um **DVD** na locadora.
`#include <DVD.hpp>`

Membros Públicos

DVD (int codigo, std::string titulo, int quantidade, std::string categoria)
*Construtor da classe **DVD**.*

virtual std::string **getTipo** () override
*Obtém o tipo do filme (**DVD**).*

virtual double **calcularValorLocacao** (int dias) override
*Calcula o valor da locação com base nos dias e na categoria do **DVD**.*

DVD (int codigo, std::string titulo, int quantidade, std::string categoria)
*Construtor da classe **DVD**.*

virtual std::string **getTipo** () override
Obtém o tipo do filme (no caso, "DVD").

virtual double **calcularValorLocacao** (int dias) override
*Calcula o valor da locação do **DVD**.*

Membros Públicos herdados de Filme

Filme (int codigo, std::string titulo, int quantidade)
*Construtor da classe **Filme**.*

virtual **~Filme** ()
*Destrutor virtual da classe **Filme**.*

int **getCodigo** ()
Obtém o código do filme.

std::string **getTitulo** ()
Obtém o título do filme.

int **getQuantidade** ()
Obtém a quantidade de cópias disponíveis do filme.

void **diminuirQuantidade** ()
Diminui a quantidade de cópias disponíveis do filme.

void **aumentarQuantidade** ()
Aumenta a quantidade de cópias disponíveis do filme.

Descrição detalhada

Classe que representa um **DVD** na locadora.

Esta classe herda da classe **Filme**.

Esta classe herda de **Filme**.

Construtores e Destrutores

DVD::DVD (int *codigo*, std::string *titulo*, int *quantidade*, std::string *categoria*)

Construtor da classe **DVD**.

Parâmetros

codigo	Código do **DVD**.
titulo	Título do **DVD**.
quantidade	Quantidade de cópias disponíveis.
categoria	Categoria do **DVD** (Lancamento, Estoque, Promocao).

DVD::DVD (int *codigo*, std::string *titulo*, int *quantidade*, std::string *categoria*)

Construtor da classe **DVD**.

Parâmetros

codigo	Código do **DVD**.
titulo	Título do **DVD**.
quantidade	Quantidade de cópias disponíveis.
categoria	Categoria do **DVD** (Lancamento, Estoque, Promocao).

Documentação das funções

virtual double DVD::calcularValorLocacao (int *dias*) `[override], [virtual]`

Calcula o valor da locação com base nos dias e na categoria do **DVD**.

Parâmetros

dias	Número de dias para locação.

Retorna

Valor total da locação.

Implementa **Filme** (*p.10*).

virtual double DVD::calcularValorLocacao (int *dias*) `[override], [virtual]`

Calcula o valor da locação do **DVD**.

Parâmetros

dias	Número de dias de locação.

Retorna

O valor total da locação.

Implementa **Filme** (*p.10*).

virtual std::string DVD::getTipo () `[override], [virtual]`

Obtém o tipo do filme (**DVD**).

Retorna

String representando o tipo do filme.

Implementa **Filme** (*p.11*).

virtual std::string DVD::getTipo () `[override], [virtual]`

Obtém o tipo do filme (no caso, "DVD").

Retorna

Uma string representando o tipo do filme.

Implementa **Filme** (*p.11*).

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

pds2/projeto final/include/**DVD.hpp**
pds2/projeto final/src/modulos/**DVD.cpp**

Referência da Classe Filme

Classe base abstrata que representa um filme na locadora.
`#include <Filme.hpp>`

Membros Públicos

Filme (int codigo, std::string titulo, int quantidade)
*Construtor da classe **Filme**.*

virtual **~Filme** ()
*Destrutor virtual da classe **Filme**.*

int **getCodigo** ()
Obtém o código do filme.

std::string **getTitulo** ()
Obtém o título do filme.

int **getQuantidade** ()
Obtém a quantidade de cópias disponíveis do filme.

virtual std::string **getTipo** ()=0
*Obtém o tipo do filme (FITA ou **DVD**).*

virtual double **calcularValorLocacao** (int dias)=0
Calcula o valor da locação com base nos dias.

void **diminuirQuantidade** ()
Diminui a quantidade de cópias disponíveis do filme.

void **aumentarQuantidade** ()
Aumenta a quantidade de cópias disponíveis do filme.

Descrição detalhada

Classe base abstrata que representa um filme na locadora.

Esta classe possui métodos virtuais puros que devem ser implementados pelas classes derivadas (**Fita** e **DVD**).

Construtores e Destrutores

Filme::Filme (int *codigo*, std::string *titulo*, int *quantidade*)

Construtor da classe **Filme**.

Construtor para a classe **Filme**.

Parâmetros

codigo	Código do filme.
titulo	Título do filme.
quantidade	Quantidade de cópias disponíveis.
codigo	O código do filme.
titulo	O título do filme.
quantidade	A quantidade de cópias do filme em estoque.

```
14 : codigo(codigo), titulo(titulo), quantidade(quantidade) {}
```

Filme::~Filme () `[virtual]`

Destrutor virtual da classe **Filme**.

Destrutor para a classe **Filme**.

```
19 {};
```

Documentação das funções

void Filme::aumentarQuantidade ()

Aumenta a quantidade de cópias disponíveis do filme.

Aumenta a quantidade de cópias do filme em um.

```
60 {
61     quantidade++;
62 }
```

virtual double Filme::calcularValorLocacao (int *dias*) `[pure virtual]`

Calcula o valor da locação com base nos dias.

Parâmetros

dias	Número de dias para locação.

Retorna

Valor total da locação.

Implementado por **DVD** (*p.7*), **Fita** (*p.13*) e **DVD** (*p.7*).

void Filme::diminuirQuantidade ()

Diminui a quantidade de cópias disponíveis do filme.

Diminui a quantidade de cópias do filme em um.

```
52 {
53     quantidade--;
54 }
```

int Filme::getCodigo ()

Obtém o código do filme.

Retorna

Código do filme.

O código do filme.

```
26 {
27     return codigo;
28 }
```

int Filme::getQuantidade ()

Obtém a quantidade de cópias disponíveis do filme.

Obtém a quantidade de cópias do filme em estoque.

Retorna

Quantidade de cópias disponíveis.

A quantidade de cópias do filme em estoque.

```
44 {
45     return quantidade;
46 }
```

virtual std::string Filme::getTipo () [pure virtual]

Obtém o tipo do filme (FITA ou **DVD**).

Retorna

String representando o tipo do filme.

Implementado por **DVD** (*p.8*), **Fita** (*p.13*) e **DVD** (*p.8*).

std::string Filme::getTitulo ()

Obtém o título do filme.

Retorna

Título do filme.

O título do filme.

```
35 {
36     return titulo;
37 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

pds2/projeto final/include/**Filme.hpp**
pds2/projeto final/src/modulos/**Filme.cpp**

Referência da Classe Fita

Classe derivada que representa uma fita de vídeo na locadora.
`#include <Fita.hpp>`

Membros Públicos

Fita (int codigo, std::string titulo, int quantidade, bool rebobinada)
*Construtor da classe **Fita**.*

virtual std::string **getTipo** () override
Obtém o tipo da fita (FITA).

virtual double **calcularValorLocacao** (int dias) override
Calcula o valor da locação com base nos dias e na condição de rebobinamento.

Membros Públicos herdados de Filme

Filme (int codigo, std::string titulo, int quantidade)
*Construtor da classe **Filme**.*

virtual **~Filme** ()
*Destrutor virtual da classe **Filme**.*

int **getCodigo** ()
Obtém o código do filme.

std::string **getTitulo** ()
Obtém o título do filme.

int **getQuantidade** ()
Obtém a quantidade de cópias disponíveis do filme.

void **diminuirQuantidade** ()
Diminui a quantidade de cópias disponíveis do filme.

void **aumentarQuantidade** ()
Aumenta a quantidade de cópias disponíveis do filme.

Descrição detalhada

Classe derivada que representa uma fita de vídeo na locadora.

Construtores e Destrutores

Fita::Fita (int *codigo*, std::string *titulo*, int *quantidade*, bool *rebobinada*)

Construtor da classe **Fita**.

Construtor para a classe **Fita**.

Parâmetros

codigo	Código da fita.
titulo	Título da fita.
quantidade	Quantidade de cópias disponíveis.
rebobinada	Indica se a fita está rebobinada.
codigo	O código da fita.
titulo	O título da fita.
quantidade	A quantidade de cópias da fita em estoque.
rebobinada	Indica se a fita está rebobinada ou não.

```
15 : Filme(codigo, titulo, quantidade), rebobinada(rebobinada) {}
```

Documentação das funções

double Fita::calcularValorLocacao (int *dias*) [override], [virtual]

Calcula o valor da locação com base nos dias e na condição de rebobinamento.

Calcula o valor da locação da fita para um determinado número de dias.

Parâmetros

dias	Número de dias para locação.

Retorna

Valor total da locação.

Parâmetros

dias	O número de dias para os quais a fita será alugada.

Retorna

O valor da locação calculado.

Implementa **Filme** (*p.10*).

```
32 {
33     double valor = 5.0;
34 
35     if (!rebobinada)
36     {
37         valor += 2.0; // Multa por não rebobinar
38     }
39 
40     return valor;
41 }
```

std::string Fita::getTipo () [override], [virtual]

Obtém o tipo da fita (FITA).

Obtém o tipo da fita.

Retorna

String representando o tipo da fita.

Uma string indicando o tipo da fita (neste caso, "FITA").

Implementa **Filme** (*p.11*).

```
22 {
23     return "FITA";
24 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

pds2/projeto final/include/**Fita.hpp**
pds2/projeto final/src/modulos/**Fita.cpp**

Referência da Classe Locadora

Classe que representa uma locadora de filmes.
`#include <Locadora.hpp>`

Membros Públicos

Locadora ()
*Construtor padrão da classe **Locadora**.*

void **lerArquivoCadastro** (std::string nomeArquivo)
Lê o cadastro de filmes a partir de um arquivo.

void **cadastrarFilmeFita** (int quantidade, int codigo, std::string titulo)
Cadastra uma fita no estoque da locadora.

void **cadastrarFilmeDVD** (int quantidade, int codigo, std::string titulo, std::string categoria)
*Cadastra um **DVD** no estoque da locadora.*

void **removerFilme** (int codigo)
Remove um filme do estoque da locadora.

void **listarFilmes** (char ordenacao)
Lista os filmes disponíveis no estoque.

void **cadastrarCliente** (std::string cpf, std::string nome)
Cadastra um novo cliente na locadora.

void **removerCliente** (std::string cpf)
Remove um cliente da locadora.

void **listarClientes** (char ordenacao)
Lista os clientes cadastrados.

void **alugarFilme** (std::string cpf)
Realiza a locação de um filme para um cliente.

void **devolverFilme** (std::string cpf, int dias)
Devolve um filme locado por um cliente.

double **calcularTotalPagar** (std::vector< int > locacoes, int dias)
Calcula o total a ser pago por um cliente com base nas locações em curso.

void **relatorioAlugueis** ()
Gera um relatório com informações sobre os aluguéis em curso.

Descrição detalhada

Classe que representa uma locadora de filmes.

Construtores e Destrutores

Locadora::Locadora ()

Construtor padrão da classe **Locadora**.

```
18 {}
```

Documentação das funções

void Locadora::alugarFilme (std::string *cpf*)

Realiza a locação de um filme para um cliente.

Realiza o aluguel de filmes para um cliente.

Parâmetros

cpf	CPF do cliente.
cpf	O CPF do cliente.

Exceções

std::runtime_error	se houver erros durante o processo de aluguel.

```
345 {
346     try
347     {
348         Cliente *cliente = buscarCliente(cpf);
349         if (cliente != nullptr)
350         {
351             if (!clientePossuiLocacao(cpf))
352             {
353                 std::vector<int> codigosFilmes;
354                 int codigo;
355 
356                 std::cout << "Cliente " << cpf << " alugou os filmes:\n";
357 
358                 while (std::cin >> codigo && codigo != -1)
359                 {
360                     Filme *filme = buscarFilme(codigo);
361                     try
362                     {
363                         if (filme != nullptr)
364                         {
365                             try
366                             {
367                                 if (filme->getQuantidade() > 0)
368                                 { // Verifica se há filmes disponíveis
369                                     codigosFilmes.push_back(codigo);
370                                     filme->diminuirQuantidade(); // Reduz o
estoque do filme
371                                     std::cout << filme->getCodigo() << " " <<
filme->getTitulo() << " " << filme->getTipo() << "\n";
372                                 }
373                                 else
374                                 {
375                                     throw std::runtime_error("ERRO: Filme " +
std::to_string(codigo) + " sem estoque disponível.");
376                                 }
377                             }
378                             catch (const std::runtime_error &ex)
379                             {
380                                 std::cerr << ex.what() << std::endl;
381                             }
```

```
382                         }
383                         else
384                         {
385                             throw std::runtime_error("ERRO: Filme " +
std::to_string(codigo) + " inexistente.");
386                         }
387                     }
388                     catch (const std::runtime_error &ex)
389                     {
390                         std::cerr << ex.what() << std::endl;
391                     }
392                 }
393 
394                 locacoesEmCurso[cpf] = codigosFilmes;
395             }
396             else
397             {
398                 throw std::runtime_error("ERRO: Cliente " + cpf + " já possui
uma locação em curso.");
399             }
400         }
401         else
402         {
403             throw std::runtime_error("ERRO: CPF inexistente.");
404         }
405     }
406     catch (const std::exception &ex)
407     {
408         std::cerr << ex.what() << std::endl;
409     }
410 }
```

void Locadora::cadastrarCliente (std::string *cpf*, std::string *nome*)

Cadastra um novo cliente na locadora.

Cadastra um cliente na locadora.

Parâmetros

cpf	CPF do cliente.
nome	Nome do cliente.
cpf	O CPF do cliente.
nome	O nome do cliente.

Exceções

std::runtime_error	se houver erros durante o cadastro.

```
250 {
251     try
252     {
253         if (validarCPF(cpf))
254         {
255             try
256             {
257                 if (buscarCliente(cpf) == nullptr)
258                 {
259                     clientes.push_back(new Cliente(cpf, nome));
260                     std::cout << "Cliente " << cpf << " cadastrado com
sucesso.\n";
261                 }
262                 else
263                 {
264                     throw std::runtime_error("CPF " + cpf + " repetido.");
265                 }
266             }
267             catch (const std::runtime_error &ex)
268             {
269                 std::cerr << ex.what() << std::endl;
270             }
271         }
272         else
273         {
274             throw std::runtime_error("CPF " + cpf + " inválido.");
275         }
```

```
276     }
277     catch (const std::exception &ex)
278     {
279         std::cerr << ex.what() << std::endl;
280     }
281 }
```

void Locadora::cadastrarFilmeDVD (int *quantidade*, int *codigo*, std::string *titulo*, std::string *categoria*)

Cadastra um **DVD** no estoque da locadora.

Cadastra um **DVD** na locadora.

Parâmetros

quantidade	Quantidade de cópias disponíveis.
codigo	Código do **DVD**.
titulo	Título do **DVD**.
categoria	Categoria do **DVD**.
quantidade	A quantidade de cópias do filme em estoque.
codigo	O código do filme.
titulo	O título do filme.
categoria	A categoria do **DVD**.

Exceções

std::runtime_error	se houver erros durante o cadastro.

```
154 {
155     try
156     {
157         if (Locadora::buscarFilme(codigo) == nullptr)
158         {
159             try
160             {
161                 if (categoria == "Lancamento" || categoria == "Promocao" ||
categoria == "Estoque")
162                 {
163                     estoqueFilmes.push_back(new DVD(codigo, titulo,
quantidade, categoria));
164                     std::cout << "Filme " << codigo << " cadastrado com
sucesso.\n";
165                 }
166                 else
167                 {
168                     throw std::runtime_error("ERRO: categoria (" + categoria +
") invalida.\nCategorias possiveis: Lancamento, Promocao e Estoque.");
169                 }
170             }
171             catch (const std::exception &ex)
172             {
173                 std::cerr << ex.what() << std::endl;
174             }
175         }
176         else
177         {
178             throw std::runtime_error("ERRO: codigo (" + std::to_string(codigo)
+ ") repetido");
179         }
180     }
181     catch (const std::exception &ex)
182     {
183         std::cerr << ex.what() << std::endl;
184     }
185 }
```

void Locadora::cadastrarFilmeFita (int *quantidade*, int *codigo*, std::string *titulo*)

Cadastra uma fita no estoque da locadora.

Cadastra um filme em fita na locadora.

Parâmetros

quantidade	Quantidade de cópias disponíveis.
codigo	Código da fita.
titulo	Título da fita.
quantidade	A quantidade de cópias do filme em estoque.
codigo	O código do filme.
titulo	O título do filme.

Exceções

std::runtime_error	se houver erros durante o cadastro.

```
124 {
125     try
126     {
127         if (Locadora::buscarFilme(codigo) == nullptr)
128         {
129             // Gera aleatoriamente um número 0 ou 1 para indicar se a fita
está rebobinada
130             static auto gen = std::bind(std::uniform_int_distribution<>(0, 1),
std::default_random_engine());
131             estoqueFilmes.push_back(new Fita(codigo, titulo, quantidade,
gen()));
132             std::cout << "Filme " << codigo << " cadastrado com sucesso.\n";
133         }
134         else
135         {
136             throw std::runtime_error("ERRO: codigo (" + std::to_string(codigo)
+ ") repetido");
137         }
138     }
139     catch (const std::exception &ex)
140     {
141         std::cerr << ex.what() << std::endl;
142     }
143 }
```

double Locadora::calcularTotalPagar (std::vector< int > *locacoes*, int *dias*)

Calcula o total a ser pago por um cliente com base nas locações em curso.

Calcula o total a ser pago pelo cliente pela locação dos filmes.

Parâmetros

locacoes	Vetor de códigos de filmes locados.
dias	Número de dias para locação.

Retorna

Valor total a ser pago.

Parâmetros

locacoes	Vetor de códigos dos filmes locados.
dias	Número de dias que os filmes foram locados.

Retorna

O total a ser pago pelo cliente.

```
532                                                                    {
533     double totalPagar = 0.0;
534
535     // Calcula o valor total a ser pago somando o valor de cada filme locado
536     for (int codigo : locacoes) {
537         Filme *filme = buscarFilme(codigo);
538         if (filme != nullptr) {
539             totalPagar += filme->calcularValorLocacao(dias);
540         }
541     }
542
543     return totalPagar;
544 }
```

void Locadora::devolverFilme (std::string *cpf*, int *dias*)

Devolve um filme locado por um cliente.

Realiza a devolução de filmes por parte do cliente.

Parâmetros

cpf	CPF do cliente.
dias	Número de dias que o filme foi locado.
cpf	O CPF do cliente.
dias	O número de dias que os filmes foram locados.

```
477                                                     {
478     try {
479         // Busca o cliente pelo CPF
480         Cliente *cliente = buscarCliente(cpf);
481 
482         // Verifica se o cliente existe
483         if (cliente != nullptr) {
484             try {
485                 // Verifica se o cliente possui uma locação em curso
486                 if (clientePossuiLocacao(cpf)) {
487                     std::cout << "Cliente " << cpf << " devolveu os
filmes:\n";
488 
489                     // Exibe os códigos dos filmes e o valor a ser pago por
cada um
490                     for (int codigo : locacoesEmCurso[cpf]) {
491                         Filme *filme = buscarFilme(codigo);
492                         if (filme != nullptr) {
493                             std::cout << codigo << " [" << filme-
>calcularValorLocacao(dias) << "]\n";
494                         }
495                     }
496 
497                     // Calcula e exibe o total a ser pago pelo cliente
498                     double totalPagar =
calcularTotalPagar(locacoesEmCurso[cpf], dias);
499                     std::cout << "Total a pagar: [" << totalPagar << "]\n";
500 
501                     // Adiciona os filmes devolvidos de volta ao estoque
502                     for (int codigo : locacoesEmCurso[cpf]) {
503                         Filme *filme = buscarFilme(codigo);
504                         if (filme != nullptr) {
505                             filme->aumentarQuantidade();
506                         }
507                     }
508 
509                     // Remove a locação em curso do cliente
510                     locacoesEmCurso.erase(cpf);
511                 } else {
512                     throw std::runtime_error("ERRO: Cliente " + cpf + " não
possui locação em curso.");
513                 }
514             } catch (const std::runtime_error &ex) {
515                 std::cerr << ex.what() << std::endl;
516             }
517         } else {
518             throw std::runtime_error("ERRO: CPF inexistente.");
519         }
520     } catch (const std::exception &ex) {
521         std::cerr << ex.what() << std::endl;
522     }
523 }
```

void Locadora::lerArquivoCadastro (std::string *nomeArquivo*)

Lê o cadastro de filmes a partir de um arquivo.

Lê um arquivo de cadastro e realiza o cadastro de filmes na locadora.

Parâmetros

nomeArquivo	Nome do arquivo de cadastro.
nomeArquivo	O nome do arquivo de cadastro.

Exceções

std::runtime_error	se o arquivo não puder ser aberto ou se ocorrerem erros durante a leitura.

```
80 {
81     // Abre o arquivo
82     std::ifstream arquivo(nomeArquivo.c_str(), std::ifstream::in);
83     try
84     {
85         if (!arquivo.is_open())
86         {
87             throw std::runtime_error("ERRO: arquivo (" + nomeArquivo + ")
inexistente.");
88         }
89         else
90         {
91             int quantidade, codigo;
92             char tipo;
93             std::string titulo, categoria;
94 
95             while (arquivo >> tipo >> quantidade >> codigo >> titulo)
96             {
97                 if (tipo == 'F')
98                 {
99                     cadastrarFilmeFita(quantidade, codigo, titulo);
100                 }
101                 else if (tipo == 'D')
102                 {
103                     arquivo >> categoria;
104                     cadastrarFilmeDVD(quantidade, codigo, titulo, categoria);
105                 }
106             }
107             arquivo.close();
108         }
109     }
110     catch (const std::exception &ex)
111     {
112         std::cerr << ex.what() << std::endl;
113     }
114 }
```

void Locadora::listarClientes (char *ordenacao*)

Lista os clientes cadastrados.

Lista os clientes da locadora.

Parâmetros

ordenacao	Caractere indicando a ordenação desejada ('N' para nome, 'C' para CPF).
ordenacao	Tipo de ordenação ('C' para CPF, 'N' para nome).

```
317 {
318     std::vector<Cliente *> clientesOrdenados = clientes;
319 
320     if (ordenacao == 'C')
321     {
322         std::sort(clientesOrdenados.begin(), clientesOrdenados.end(),
323                   [](Cliente *a, Cliente *b)
324                   { return a->getCPF() < b->getCPF(); });
325     }
326     else if (ordenacao == 'N')
327     {
328         std::sort(clientesOrdenados.begin(), clientesOrdenados.end(),
329                   [](Cliente *a, Cliente *b)
330                   { return a->getNome() < b->getNome(); });
331     }
332 
333     for (auto cliente : clientesOrdenados)
334     {
335         std::cout << cliente->getCPF() << " " << cliente->getNome() << "\n";
336     }
```

```
337 }
```

void Locadora::listarFilmes (char *ordenacao*)

Lista os filmes disponíveis no estoque.

Lista os filmes do estoque da locadora.

Parâmetros

ordenacao	Caractere indicando a ordenação desejada ('T' para título, 'C' para código).
ordenacao	Tipo de ordenação ('C' para código, 'T' para título).

```
220 {
221     std::vector<Filme *> filmesOrdenados = estoqueFilmes;
222 
223     if (ordenacao == 'C')
224     {
225         std::sort(filmesOrdenados.begin(), filmesOrdenados.end(),
226                   [](Filme *a, Filme *b)
227                   { return a->getCodigo() < b->getCodigo(); });
228     }
229     else if (ordenacao == 'T')
230     {
231         std::sort(filmesOrdenados.begin(), filmesOrdenados.end(),
232                   [](Filme *a, Filme *b)
233                   { return a->getTitulo() < b->getTitulo(); });
234     }
235 
236     for (auto filme : filmesOrdenados)
237     {
238         std::cout << filme->getCodigo() << " " << filme->getTitulo() << " " <<
filme->getQuantidade()
239                   << " " << filme->getTipo() << "\n";
240     }
241 }
```

void Locadora::relatorioAlugueis ()

Gera um relatório com informações sobre os aluguéis em curso.

Gera um relatório de todas as locações em curso.

```
587                                 {
588     std::string nome;
589 
590     std::unordered_map<std::string, std::vector<int>>::iterator it;
591     std::cout << "-----------------------------------------------------" <<
std::endl;
592     std::cout << "Relatorio de alugueis" << std::endl;
593     std::cout << std::endl;
594 
595     // Itera sobre as locações em curso e exibe as informações
596     for (it = locacoesEmCurso.begin(); it != locacoesEmCurso.end(); it++) {
597         nome = Locadora::buscarCliente(it->first)->getNome();
598         std::cout << it->first << " " << nome << " ";
599         for (int codigo : it->second) {
600             std::cout << codigo << " ";
601         }
602         std::cout << std::endl;
603     }
604 
605     std::cout << "-----------------------------------------------------" <<
std::endl;
606 }
```

void Locadora::removerCliente (std::string *cpf*)

Remove um cliente da locadora.

Parâmetros

cpf	CPF do cliente a ser removido.
cpf	O CPF do cliente a ser removido.

Exceções

std::runtime_error	se o cliente não existir na locadora.

```
289 {
290     auto it = std::find_if(clientes.begin(), clientes.end(), [cpf](Cliente
*cliente)
291                             { return cliente->getCPF() == cpf; });
292 
293     try
294     {
295         if (it != clientes.end())
296         {
297             delete *it;
298             clientes.erase(it);
299             std::cout << "Cliente " << cpf << " removido com sucesso.\n";
300         }
301         else
302         {
303             throw std::runtime_error("CPF " + cpf + " não encontrado para
remoção.");
304         }
305     }
306     catch (const std::exception &ex)
307     {
308         std::cerr << ex.what() << std::endl;
309     }
310 }
```

void Locadora::removerFilme (int *codigo*)

Remove um filme do estoque da locadora.

Parâmetros

codigo	Código do filme a ser removido.
codigo	O código do filme a ser removido.

Exceções

std::runtime_error	se o filme não existir no estoque.

```
193 {
194     auto it = std::find_if(estoqueFilmes.begin(), estoqueFilmes.end(),
[codigo](Filme *filme)
195                             { return filme->getCodigo() == codigo; });
196     try
197     {
198         if (it != estoqueFilmes.end())
199         {
200             delete *it;
201             estoqueFilmes.erase(it);
202             std::cout << "Filme " << codigo << " removido com sucesso.\n";
203         }
204         else
205         {
206             throw std::runtime_error("ERRO: codigo inexistente.");
207         }
208     }
209     catch (const std::exception &ex)
210     {
211         std::cerr << ex.what() << std::endl;
212     }
213 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

pds2/projeto final/include/**Locadora.hpp**
pds2/projeto final/src/modulos/**Locadora.cpp**

Arquivos

Referência do Arquivo pds2/projeto final/include/Cliente.hpp

```
#include <string>
#include <vector>
```

Componentes

class **Cliente***Classe que representa um cliente da locadora.*

Cliente.hpp

Ir para a documentação desse arquivo.

```
1 // Cliente.hpp
2
3 #ifndef CLIENTE_HPP
4 #define CLIENTE_HPP
5
6 #include <string>
7 #include <vector>
8
12 class Cliente
13 {
14 public:
20     Cliente(std::string cpf, std::string nome);
21
26     std::string getCPF();
27
32     std::string getNome();
33
34 private:
35     std::string cpf;
36     std::string nome;
37     std::vector<int> locacoes;
38 };
39
40 #endif // CLIENTE_HPP
```

Referência do Arquivo pds2/projeto final/include/DVD.hpp

```
#include "Filme.hpp"
```

Componentes

class **DVD***Classe que representa um **DVD** na locadora.*

DVD.hpp

Ir para a documentação desse arquivo.

```
1 #ifndef DVD_HPP
2 #define DVD_HPP
3
4 #include "Filme.hpp"
5
11 class DVD : public Filme
12 {
13 public:
21     DVD(int codigo, std::string titulo, int quantidade, std::string categoria);
22
27     virtual std::string getTipo() override;
28
34     virtual double calcularValorLocacao(int dias) override;
35
36 private:
37     std::string categoria;
38 };
39
40 #endif // DVD_HPP
```

Referência do Arquivo pds2/projeto final/include/Filme.hpp

`#include <string>`

Componentes

class **Filme***Classe base abstrata que representa um filme na locadora.*

Filme.hpp

Ir para a documentação desse arquivo.

```
1 #ifndef FILME_HPP
2 #define FILME_HPP
3
4 #include <string>
5
12 class Filme
13 {
14 public:
21     Filme(int codigo, std::string titulo, int quantidade);
22
26     virtual ~Filme();
27
32     int getCodigo();
33
38     std::string getTitulo();
39
44     int getQuantidade();
45
50     virtual std::string getTipo() = 0;
51
57     virtual double calcularValorLocacao(int dias) = 0;
58
62     void diminuirQuantidade();
63
67     void aumentarQuantidade();
68
69 private:
70     int codigo;
71     std::string titulo;
72     int quantidade;
73 };
74
75 #endif // FILME_HPP
```

Filme.hpp

Ir para a documentação desse arquivo.

```
1 #ifndef FILME_HPP
2 #define FILME_HPP
3
4 #include <string>
5
12 class Filme
13 {
14 public:
21     Filme(int codigo, std::string titulo, int quantidade);
22
26     virtual ~Filme();
27
32     int getCodigo();
33
38     std::string getTitulo();
39
44     int getQuantidade();
45
50     virtual std::string getTipo() = 0;
51
57     virtual double calcularValorLocacao(int dias) = 0;
58
62     void diminuirQuantidade();
63
67     void aumentarQuantidade();
68
69 private:
70     int codigo;
71     std::string titulo;
72     int quantidade;
73 };
74
75 #endif // FILME_HPP
```

Referência do Arquivo pds2/projeto final/include/Fita.hpp

```
#include <random>
#include <algorithm>
#include "Filme.hpp"
```

Componentes

class **Fita***Classe derivada que representa uma fita de vídeo na locadora.*

Fita.hpp

Ir para a documentação desse arquivo.

```
1 #ifndef FITA_HPP
2 #define FITA_HPP
3
4 #include <random>
5 #include <algorithm>
6 #include "Filme.hpp"
7
11 class Fita : public Filme
12 {
13 public:
21     Fita(int codigo, std::string titulo, int quantidade, bool rebobinada);
22
27     virtual std::string getTipo() override;
28
34     virtual double calcularValorLocacao(int dias) override;
35
36 private:
37     bool rebobinada;
38 };
39
40 #endif // FITA_HPP
```

Referência do Arquivo pds2/projeto final/include/Locadora.hpp

```
#include <vector>
#include <unordered_map>
#include <algorithm>
#include "Cliente.hpp"
#include "Filme.hpp"
```

Componentes

class **Locadora***Classe que representa uma locadora de filmes.*

Locadora.hpp

Ir para a documentação desse arquivo.

```
1 #ifndef LOCADORA_HPP
2 #define LOCADORA_HPP
3
4 #include <vector>
5 #include <unordered_map>
6 #include <algorithm>
7 #include "Cliente.hpp"
8 #include "Filme.hpp"
9
13 class Locadora {
14 public:
18     Locadora();
19
24     void lerArquivoCadastro(std::string nomeArquivo);
25
32     void cadastrarFilmeFita(int quantidade, int codigo, std::string titulo);
33
41     void cadastrarFilmeDVD(int quantidade, int codigo, std::string titulo,
std::string categoria);
42
47     void removerFilme(int codigo);
48
53     void listarFilmes(char ordenacao);
54
60     void cadastrarCliente(std::string cpf, std::string nome);
61
66     void removerCliente(std::string cpf);
67
72     void listarClientes(char ordenacao);
73
78     void alugarFilme(std::string cpf);
79
85     void devolverFilme(std::string cpf, int dias);
86
93     double calcularTotalPagar(std::vector<int> locacoes, int dias);
94
98     void relatorioAlugueis();
99
100 private:
101     std::vector<Filme *> estoqueFilmes;
102     std::vector<Cliente *> clientes;
103     std::unordered_map<std::string, std::vector<int>> locacoesEmCurso;
104
110     Filme* buscarFilme(int codigo);
111
117     Cliente* buscarCliente(std::string cpf);
118
124     bool clientePossuiLocacao(std::string cpf);
125
131     bool validarCPF(std::string cpf);
132 };
133
134 #endif // LOCADORA_HPP
```

Referência do Arquivo pds2/projeto final/src/main.cpp

```
#include <iostream>
#include <stdexcept>
#include "Locadora.hpp"
```

Funções

int **main** ()

Função principal que executa o sistema da locadora.

Funções

int main ()

Função principal que executa o sistema da locadora.

Cria uma instância da classe **Locadora**, exibe o menu de comandos e executa os comandos correspondentes até que o usuário escolha sair.

Retorna

0 para indicar que o programa foi executado com sucesso.

```
16 {
17     Locadora locadora;
18     std::string comando;
19     std::cout << "------------------------LOCADORA--------------------------"
<< std::endl;
20     std::cout << "Bem vindo!" << std::endl;
21     std::cout << "Abaixo está o menu de comandos para a utilização da" <<
std::endl;
22     std::cout << "ferramenta:" << std::endl;
23     std::cout << std::endl;
24     std::cout << "MENU" << std::endl;
25     std::cout << "LA: ler arquivo de estoque" << std::endl;
26     std::cout << "CF: cadastrar filme no sistema" << std::endl;
27     std::cout << "RF: remover filme no sistema" << std::endl;
28     std::cout << "LF: listar filmes no sistema" << std::endl;
29     std::cout << "CC: cadastrar cliente no sistema" << std::endl;
30     std::cout << "RC: remover cliente no sistema" << std::endl;
31     std::cout << "LC: listar clientes no sistema" << std::endl;
32     std::cout << "AL: registar um aluguel" << std::endl;
33     std::cout << "DV: registar uma devolucao" << std::endl;
34     std::cout << "RA: exibir relatorio de alugueis" << std::endl;
35     std::cout << "HELP: exibe detalhes dos comandos" << std::endl;
36     std::cout << "FS: finalizar o sistema" << std::endl;
37     std::cout << "-----------------------------------------------------" <<
std::endl;
38     std::cout << std::endl;
39 
40     int dias;
41 
42     while (std::cin >> comando && comando != "FS")
43     {
44         try
45         {
46             if (comando == "LA")
47             {
48                 std::string nomeArquivo;
49                 std::getline(std::cin, nomeArquivo);
50                 // Remove o espaço (primeiro caractere) da string
51                 nomeArquivo.erase(0, 1);
52                 locadora.lerArquivoCadastro(nomeArquivo);
53             }
54             else if (comando == "CF")
55             {
```

```
56                  char tipo;
57                  int quantidade, codigo;
58                  std::string titulo, categoria;
59
60                  std::cin >> tipo >> quantidade >> codigo >> titulo;
61                  if (tipo == 'F')
62                  {
63                      locadora.cadastrarFilmeFita(quantidade, codigo, titulo);
64                  }
65                  else if (tipo == 'D')
66                  {
67                      std::cin >> categoria;
68                      locadora.cadastrarFilmeDVD(quantidade, codigo, titulo,
categoria);
69                  }
70              }
71              else if (comando == "RF")
72              {
73                  int codigo;
74                  std::cin >> codigo;
75                  locadora.removerFilme(codigo);
76              }
77              else if (comando == "LF")
78              {
79                  char ordenacao;
80                  std::cin >> ordenacao;
81                  locadora.listarFilmes(ordenacao);
82              }
83              else if (comando == "CC")
84              {
85                  std::string cpf, nome;
86                  std::cin >> cpf >> nome;
87                  locadora.cadastrarCliente(cpf, nome);
88              }
89              else if (comando == "RC")
90              {
91                  std::string cpf;
92                  std::cin >> cpf;
93                  locadora.removerCliente(cpf);
94              }
95              else if (comando == "LC")
96              {
97                  char ordenacao;
98                  std::cin >> ordenacao;
99                  locadora.listarClientes(ordenacao);
100             }
101             else if (comando == "AL")
102             {
103                 std::string cpf;
104                 std::cin >> cpf;
105                 locadora.alugarFilme(cpf);
106             }
107             else if (comando == "DV")
108             {
109                 std::string cpf;
110                 std::cin >> cpf >> dias;
111                 locadora.devolverFilme(cpf, dias);
112             }
113             else if (comando == "RA")
114             {
115                 locadora.relatorioAlugueis();
116             }
117             else if (comando == "HELP")
118             {
119                 std::cout << "MENU HELP--------------------------------------"
<< std::endl;
120                 std::cout << "LA <Nome do Arquivo> (caminho de pastas
completo)" << std::endl;
121                 // ... (continuação da exibição detalhada dos comandos)
122                 std::cout << "FS (finalizar o sistema)" << std::endl;
123                 std::cout << "-----------------------------------------------
" << std::endl;
124             }
125             else
126             {
127                 throw std::invalid_argument("Erro: Comando inválido. Digite
HELP para ver os comandos disponíveis");
```

```
128             }
129         }
130         catch (const std::exception &e)
131         {
132             std::cout << e.what() << "\n";
133         }
134 
135         std::cout << std::endl;
136         std::cout << "Digite um comando (ou 'FS' para sair): " << std::endl;
137     }
138 
139     return 0;
140 }
```

Referência do Arquivo pds2/projeto final/src/mainTests.cpp

```
#include "doctest.h"
```

Funções

int **main** ()

Funções

int main ()

```
4 {
5 
6     return 0;
7 }
```

Referência do Arquivo pds2/projeto final/src/modulos/Cliente.cpp

```
#include <string>
#include <vector>
```

Componentes

class **Cliente***Classe que representa um cliente da locadora.*

Definições e Macros

#define **CLIENTE_HPP**

Definições e macros

#define CLIENTE_HPP

Referência do Arquivo pds2/projeto final/src/modulos/DVD.cpp

```
#include "Filme.hpp"
```

Componentes

class **DVD***Classe que representa um **DVD** na locadora.*

Definições e Macros

#define **DVD_HPP**

Definições e macros

#define DVD_HPP

Referência do Arquivo pds2/projeto final/src/modulos/Filme.cpp

Arquivo de implementação para a classe **Filme**.

```
#include "Filme.hpp"
```

Descrição detalhada

Arquivo de implementação para a classe **Filme**.

Referência do Arquivo pds2/projeto final/src/modulos/Fita.cpp

Implementação da classe **Fita**, derivada da classe **Filme**.

```
#include "Fita.hpp"
```

Descrição detalhada

Implementação da classe **Fita**, derivada da classe **Filme**.

Referência do Arquivo pds2/projeto final/src/modulos/Locadora.cpp

Implementação da classe **Locadora**.

```
#include "Locadora.hpp"
#include "Fita.hpp"
#include "DVD.hpp"
#include <fstream>
#include <algorithm>
#include <iostream>
```

Descrição detalhada

Implementação da classe **Locadora**.

Referência do Arquivo pds2/projeto final/tests/testeCliente.cpp

```
#include "doctest.h"
#include "Cliente.hpp"
```

Definições e Macros

#define **DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN**

Funções

TEST_CASE ("Teste Cliente")

Definições e macros

#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

Funções

TEST_CASE ("Teste Cliente")

```
6 {
7     Cliente teste = Cliente("12590359640", "Milton");
8     CHECK(teste.getCPF() == "12590359640");
9     CHECK(teste.getNome() == "Milton");
10 }
```

Referência do Arquivo pds2/projeto final/tests/testeDVD.cpp

```
#include "doctest.h"
#include "DVD.hpp"
```

Funções

TEST_CASE ("Teste **DVD** - Geral")
TEST_CASE ("Teste **DVD** - Dia zerado")

Funções

TEST_CASE ("Teste DVD - Dia zerado")

```
15 {
16     DVD teste = DVD(99, "FilmeTeste", 3, "Promocao");
17     DVD teste2 = DVD(98, "FilmeTeste", 3, "Lancamento");
18     DVD teste3 = DVD(97, "FilmeTeste", 3, "Estoque");
19     CHECK(teste.calcularValorLocacao(0) == 10);
20     CHECK(teste2.calcularValorLocacao(0) == 20);
21     CHECK(teste3.calcularValorLocacao(0) == 10);
22 }
```

TEST_CASE ("Teste DVD - Geral")

```
5 {
6     DVD teste = DVD(99, "FilmeTeste", 3, "Promocao");
7     DVD teste2 = DVD(98, "FilmeTeste", 3, "Lancamento");
8     DVD teste3 = DVD(97, "FilmeTeste", 3, "Estoque");
9     CHECK(teste.getTipo() == "DVD");
10     CHECK(teste.calcularValorLocacao(3) == 10);
11     CHECK(teste2.calcularValorLocacao(3) == 60);
12     CHECK(teste3.calcularValorLocacao(3) == 30);
13 }
```

Referência do Arquivo pds2/projeto final/tests/testeFilme.cpp

```
#include "doctest.h"
#include "Fita.hpp"
```

Funções

TEST_CASE ("Teste **Filme** - Geral")

Funções

TEST_CASE ("Teste Filme - Geral")

```
5 {
6     Fita teste = Fita(99, "FilmeTeste", 3, true);
7     CHECK(teste.getCodigo() == 99);
8     CHECK(teste.getTitulo() == "FilmeTeste");
9     CHECK(teste.getQuantidade() == 3);
10     teste.diminuirQuantidade();
11     teste.diminuirQuantidade();
12     CHECK(teste.getQuantidade() == 1);
13     teste.aumentarQuantidade();
14     CHECK(teste.getQuantidade() == 2);
15 }
```

Referência do Arquivo pds2/projeto final/tests/testeFita.cpp

```
#include "doctest.h"
#include "FITA.hpp"
```

Funções

TEST_CASE ("Teste Fita - Geral")

Funções

TEST_CASE ("Teste Fita - Geral")

```
5 {
6     Fita teste = Fita(99, "FilmeTeste", 3, true);
7     Fita teste2 = Fita(98, "FilmeTeste", 3, false);
8     CHECK(teste.getTipo() == "FITA");
9     CHECK(teste.calcularValorLocacao(0) == 5);
10     CHECK(teste2.calcularValorLocacao(3) == 7);
11 }
```