Versão 1.0

# Projeto Final

Projeto Final PDS2 - UFMG Integrantes: Artur Xavier Nascimento, Pedro Bacelar Rigueira, Caio Souza Grossi, Luiz Eduardo Gonçalves da Silveira e Artur Fonseca de Souza

## Apresentação do problema

O programa em questão aborda a gestão de uma locadora de filmes, sendo necessário implementar um sistema capaz de controlar o estoque de filmes, cadastrar e remover clientes, realizar operações de aluguel e devolução, além de imprimir registros de clientes e filmes. O sistema também deve permitir a leitura de informações de um arquivo de estoque para facilitar o carregamento inicial de filmes.

## Visão Geral da Solução:

A nossa solução é estruturada em classes, sendo as principais classes **Filme**, **DVD**, **Fita**, **Cliente** e **Locadora**. Assim sendo, a classe abstrata **Filme** serve como base para as classes específicas **DVD** e **Fita**, que herdam características comuns e implementam métodos específicos e, a classe **Locadora** coordena as operações entre clientes e o estoque de filmes.

A leitura do arquivo de estoque é feita pela função ler\_Arquivo\_de\_Estoque, que utiliza a classe **Filme** para criar instâncias de **DVD** e **Fita** conforme o tipo de filme especificado no arquivo. As operações de aluguel e devolução são gerenciadas pelos métodos alugar\_Filme e devolver\_Filme, que atualizam o estoque e registram as transações em um arquivo de log.

## Principais Dificuldades Encontradas:

Uma das principais dificuldades foi lidar com a complexidade das operações envolvendo múltiplas classes, especialmente no controle de estoque e na interação com o sistema de arquivos para a leitura do arquivo de estoque e o registro das operações de log. Além disso, a gestão dinâmica de memória para evitar vazamentos foi um ponto de atenção, especialmente no destrutor da classe **Locadora**, onde é necessário liberar os recursos alocados para clientes e filmes.

# 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 (contém as informações relacionadas aos clientes, como nome e id)**

**DVD (contém as informações relacionadas aos DVDs, como categoria, quantidade, título e id)**

**Filme (contém as informações relacionadas aos filmes, e serve de base hierárquica para as classes fita e DVD)**

**Fita (contém as informações relacionadas as fitas)**

**Locadora (classe responsável por gerir as operações da locadora)**

# Índice dos Arquivos

## Lista de Arquivos

Esta é a lista de todos os arquivos:

**TRABALHO FINAL/cliente.cpp**

**TRABALHO FINAL/cliente.hpp**

**TRABALHO FINAL/dvd.cpp**

**TRABALHO FINAL/dvd.hpp**

**TRABALHO FINAL/Filme.cpp**

**TRABALHO FINAL/Filme.hpp**

**TRABALHO FINAL/fita.cpp**

**TRABALHO FINAL/fita.hpp**

**TRABALHO FINAL/Locadora.cpp**

**TRABALHO FINAL/Locadora.hpp**

**TRABALHO FINAL/main.cpp**

# Classes

## Referência da Classe Cliente

#include <cliente.hpp>

### Membros Públicos

**Cliente** (long cpf, string nome)

Construtor da classe **Cliente**, inicializa os membros nome e cpf.

~Cliente ()

Destrutor da classe **Cliente**.

string **get\_nome** () const

long **get\_cpf** () const

### Amigos

class Locadora

### Construtores e Destrutores

#### Cliente::Cliente (long cpf, string nome)

Construtor da classe **Cliente**, inicializa os membros nome e cpf.

7 Cliente::Cliente(long cpf, string nome) : \_cpf(cpf), \_nome(nome){};

#### Cliente::~Cliente ()

Destrutor da classe **Cliente**.

9 Cliente::~Cliente(){};

### Documentação das funções

#### long Cliente::get\_cpf () const

Armazena o cpf do cliente

12 {

13 return \_cpf;

14 }

#### string Cliente::get\_nome () const

Armazena o nome do cliente

16 {

17 return \_nome;

18 }

### Documentação dos símbolos amigos e relacionados

#### friend class Locadora[friend]

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

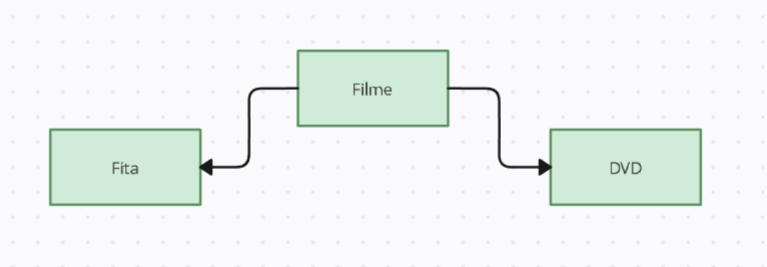
TRABALHO FINAL/**cliente.hpp**

TRABALHO FINAL/**cliente.cpp**

## Referência da Classe DVD

#include <dvd.hpp>

Diagrama de hierarquia da classe DVD:



### Membros Públicos

**DVD** (int id, string titulo, int qtd\_disp, string categoria)

Categoria do **DVD**.

~DVD ()

Destrutor da classe **DVD**.

virtual string **get\_categoria** ()

Método que retorna a categoria.

virtual string **get\_tipo** () override

Método que retorna o tipo do filme.

float **calc\_valor\_locacao** (int dias) const override

Método que calcula o valor da locação, com base na quantidade de dias.

#### Membros Públicos herdados de Filme

**Filme** (int id, string titulo, int qtd\_disp)

Identificação, título e quantidade disponível.

virtual **~Filme** ()

Construtor que realiza a iniciação e inclusão de métodos públicos de acesso.

int **get\_id** () const

**get\_id()** para obter o identificador

string **get\_titulo** () const

**get\_titulo()** para obter o titulo

int **get\_qtdDisp** () const

**get\_qtdDisp** () para obter a quantidade disponível

### Construtores e Destrutores

#### DVD::DVD (int id, string titulo, int qtd\_disp, string categoria)

Construtor da classe **DVD**, que inicializa os membros.

12 DVD::DVD(int id, string titulo, int qtd\_disp, string categoria) : Filme(id, titulo, qtd\_disp), \_categoria(categoria){};

#### DVD::~DVD ()

O destrutor realiza a limpeza quando um objeto **DVD** e removido.

14 DVD::~DVD(){};

### Documentação das funções

#### float DVD::calc\_valor\_locacao (int dias) const[override], [virtual]

Calcula o valor da locacao com base no número de dias para os quais o cliente deseja aluga-lo.

Implementa **Filme**.

22 {

23 if (\_categoria == "Lancamento")

24 {

25 return (20.0\*dias);

26 }

27 else if (\_categoria == "Estoque")

28 {

29 return (10.0\*dias);

30 }

31 else if (\_categoria == "Promocao")

32 {

33 return 10.0;

34 }

35 else

36 {

37 cout << "Erro: Categoria Invalida";

38 return 0.0;

39 }

40

41 }

#### string DVD::get\_categoria ()[virtual]

Método que armazena e retorna a categoria.

Implementa **Filme**.

16 {

17 return \_categoria;

18 }

#### string DVD::get\_tipo ()[override], [virtual]

Método que retorna o tipo do filme.

Implementa **Filme**.

20 { return "DVD"; }

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

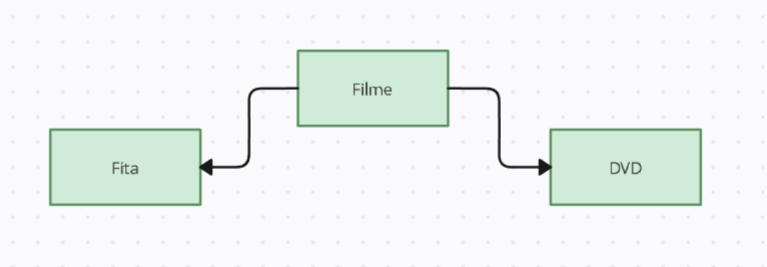
TRABALHO FINAL/**dvd.hpp**

TRABALHO FINAL/**dvd.cpp**

## Referência da Classe Filme

#include <Filme.hpp>

Diagrama de hierarquia da classe Filme:



### Membros Públicos

**Filme** (int id, string titulo, int qtd\_disp)

Identificação, título e quantidade disponível.

virtual **~Filme** ()

O destrutor virtual e declarado, indicando que a classe e abstrata e pode conter métodos virtuais puros.

virtual string **get\_categoria** ()=0

Método virtual para obter a categoria do filme.

virtual string **get\_tipo** ()=0

Método virtual para obter o tipo do filme.

int **get\_id** () const

**get\_id()** para obter o identificador.

string **get\_titulo** () const

**get\_titulo()** para obter o título.

int **get\_qtdDisp** () const

**get\_qtdDisp()** para obter a quantidade disponível.

virtual float **calc\_valor\_locacao** (int dias) const =0

**calc\_valor\_locacao** (int dias) para obter o valor de locação.

### Amigos

class Locadora

### Construtores e Destrutores

#### Filme::Filme (int id, string titulo, int qtd\_disp)

A classe possui um construtor que inicializa os atributos da instancia (\_id, \_titulo e \_qtd\_disp) com os valores fornecidos como argumentos.

18 : \_id(id), \_titulo(titulo), \_qtd\_disp(qtd\_disp){};

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

Construtor que realiza a iniciação e inclusão de métodos públicos de acesso.

20 {};

### Documentação das funções

#### virtual float Filme::calc\_valor\_locacao (int dias) const[pure virtual]

calc\_valor\_locacao (int dias) calcula o valor de locação.

Implementado por **DVD** e **Fita**.

#### virtual string Filme::get\_categoria ()[pure virtual]

get\_categoria () para obter a categoria.

Implementado por **DVD** e **Fita**.

#### int Filme::get\_id () const

Armazena o id

22 {

23 return \_id;

24 }

#### int Filme::get\_qtdDisp () const

Armazena a quantidade disponível

30 {

31 return \_qtd\_disp;

32 }

#### virtual string Filme::get\_tipo ()[pure virtual]

Implementado por **DVD** e **Fita**.

#### string Filme::get\_titulo () const

Armazena o título.

26 {

27 return \_titulo;

28 }

### Documentação dos símbolos amigos e relacionados

#### friend class Locadora[friend]

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

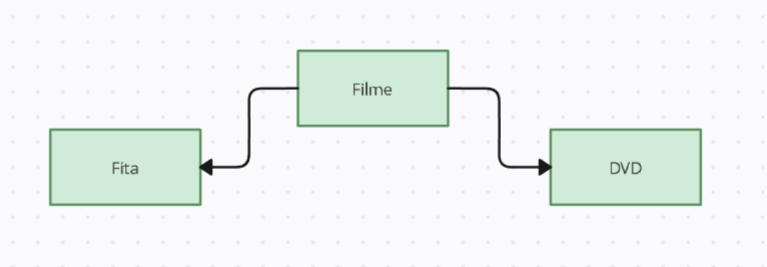
TRABALHO FINAL/**Filme.hpp**

TRABALHO FINAL/**Filme.cpp**

## Referência da Classe Fita

#include <fita.hpp>

Diagrama de hierarquia da classe Fita:



### Membros Públicos

**Fita** (int id, string titulo, int qtd\_disp)

Construtor da classe **Fita**.

~Fita ()

Destrutor da classe **Fita**.

virtual string **get\_categoria** () override

Método que retorna à categoria.

virtual string **get\_tipo** () override

Método que retorna o tipo do filme, no caso fita.

bool get\_isRebobinada () const

Método que verifica se a fita está rebobinada.

float **calc\_valor\_locacao** (int dias) const override

Método que calcula o valor da locação da fita com base na quantidade de dias.

#### Membros Públicos herdados de Filme

**Filme** (int id, string titulo, int qtd\_disp)

Construtor da classe **Filme**.

virtual **~Filme** ()

Destrutor virtual da classe **Filme**.

int **get\_id** () const

**get\_id()** para obter o identificador

string **get\_titulo** () const

**get\_titulo()** para obter o titulo

int **get\_qtdDisp** () const

**get\_qtdDisp** () para obter a quantidade disponível

### Construtores e Destrutores

#### Fita::Fita (int id, string titulo, int qtd\_disp)

O construtor inicializa os atributos da classe, herdando alguns de sua classe base **Filme** e introduzindo um novo atributo privado chamado \_isRebobinada (este atributo é randomizado no momento da criação da fita, determinando se a fita está ou não rebobinada).

12 : Filme(id, titulo, qtd\_disp){

13 \_isRebobinada = rand() % 2 == 0; //randomizo se a fita estara rebobinada ou nao

14 };

#### Fita::~Fita ()

O destrutor faz a limpeza quando um objeto **fita** é destruído.

16 {};

### Documentação das funções

#### float Fita::calc\_valor\_locacao (int dias) const[override], [virtual]

Método que calcula o valor da locação da fita com base na quantidade de dias.

Implementa **Filme**.

25 {

26 if (\_isRebobinada)

27 {

28 return 5.0;

29 }

30 else

31 {

32 return 7.0; // 5 + 2 de multa

33 }

34 };

#### string Fita::get\_categoria ()[override], [virtual]

A função **get\_categoria()** retorna uma string vazia, indicando que as fitas não possuem categorias especificas.

Implementa **Filme**.

18 { return ""; }

#### bool Fita::get\_isRebobinada () const

O método **get\_isRebobinada()** retorna o estado da fita, informando se ela está rebobinada ou não.

21 {

22 return \_isRebobinada;

23 }

#### string Fita::get\_tipo ()[override], [virtual]

Método que retorna o tipo do filme, no caso fita.

Implementa **Filme** (p.pagenum).

19 { return "FITA"; }

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

TRABALHO FINAL/**fita.hpp**

TRABALHO FINAL/**fita.cpp**

## Referência da Classe Locadora

#include <Locadora.hpp>

### Membros Públicos

**Locadora** (string nome)

Construtor da classe **Locadora** que inicializa o nome e abre o arquivo de logs.

~Locadora ()

Destrutor da classe **Locadora** que libera a memória alocada para clientes e filmes.

void **getNome** ()

Obtém o nome da locadora.

void **alugar\_Filme** (vector< int > codigos, long cpf)

Funções relacionadas a alugar e devolver filmes.

void **devolver\_Filme** (long cpf)

void **cadastrar\_cliente** (long cpf, string nome)

void **remover\_cliente** (long cpf)

void **imprimir\_clientes** (char tipo\_ordenacao)

Cliente \* buscar\_cliente (long cpf)

Funções relacionadas ao cadastro, remoção, impressão e busca de clientes.

map< int, **Filme** \* > **getEstoque** ()

void **ler\_Arquivo\_de\_Estoque** (string nome\_do\_arquivo)

void **cadastrar\_Filme** (bool file, char tipo, int quantidade, int codigo, string titulo, string categoria)

void **remover\_Filme** (int codigo)

void **imprimir\_Estoque** (char tipo\_ordenacao)

Métodos relacionados ao estoque de filmes.

### Construtores e Destrutores

#### Locadora::Locadora (string nome)

O construtor **Locadora(string nome)** inicializa os membros, incluindo o nome da locadora e o objeto de log.

30 : \_Nome(nome)

31 {

32 string nome\_arquivo\_logs = "Alugueis\_" + nome + "\_log.txt";

33 \_Logs.open( nome\_arquivo\_logs, ios::trunc);

34 }

#### Locadora::~Locadora ()

O destrutor realiza a limpeza dos recursos de forma dinâmica, tanto para clientes quanto para filmes.

37 {

38 for( auto it = \_Clientes.back(); it != \_Clientes.front(); it-- ){

39 delete it;

40 \_Clientes.pop\_back();

41 }

42 for( auto it:\_Estoque ){

43 delete it.second;

44 \_Estoque.erase( it.first );

45 }

46 \_Logs.close();

47 }

### Documentação das funções

#### void Locadora::alugar\_Filme (vector< int > codigos, long cpf)

Funções relacionadas a alugar e devolver filmes.

50 {

51 if ( this->buscar\_cliente( cpf ) == nullptr ) { return; };

52

54 vector<int> codigos\_invalidos;

55 vector<Filme\*> filmes\_alugados;

56 for( auto id\_novo\_filme:codigos )

57 {

58 if( \_Estoque.find( id\_novo\_filme ) == \_Estoque.end() )

59 {

60 codigos\_invalidos.push\_back( id\_novo\_filme );

61 }

62 else

63 {

64 filmes\_alugados.push\_back( \_Estoque.at( id\_novo\_filme ) );

65 }

66 }

67

69 if( filmes\_alugados.empty() == false ){

70

71 stringstream stream\_cliente;

72 stream\_cliente << "Cliente " << cpf << " "

73 << this->buscar\_cliente( cpf )->get\_nome()

74 << " alugou os filmes:" << endl;

75

76 string linha\_cliente;

77 getline (stream\_cliente, linha\_cliente);

78 cout << linha\_cliente << endl;

79 \_Logs << linha\_cliente << endl;

80 //

81 }

82

83 for( auto novo\_filme:filmes\_alugados )

84 {

85 if( novo\_filme->get\_qtdDisp() > 0 ){

86 novo\_filme->\_qtd\_disp -= 1;

87

88 this->buscar\_cliente( cpf )->\_Filmes\_Alugados.push\_back( novo\_filme );

89

91 stringstream stream\_linha;

92 stream\_linha << novo\_filme->get\_id()

93 << " " << novo\_filme->get\_titulo()

94 << " " << novo\_filme->get\_tipo();

95

96 string linha;

97 getline(stream\_linha, linha);

98 cout << linha << endl;

99 \_Logs << linha << endl;

100

101 }

102 else

103 cout << "ERRO: "

104 << novo\_filme->get\_id()

105 << " " << novo\_filme->get\_titulo()

106 << " " << novo\_filme-> get\_categoria() << "indisponivel" << endl;

107 }

108

109 for( auto t:codigos\_invalidos )

110 {

111 cout << "ERRO: Filme " << t << " inexistente" << endl;

112 }

113

114 }

#### Cliente \* Locadora::buscar\_cliente (long cpf)

Faz a busca de clientes por CPF

353 {

354 for( Cliente\* it:\_Clientes ){

355 if( it->get\_cpf() == cpf ) {

356 return it;

357 }

358 }

359 cout << "ERRO: CPF inexistente" << endl;

360 return nullptr;

361 }

#### void Locadora::cadastrar\_cliente (long cpf, string nome)

Funções relacionadas ao cadastro, remoção, impressão e busca de clientes.

291 {

292

293 if( any\_of(\_Clientes.begin(), \_Clientes.end(),

294 [cpf](const Cliente\* C) {return C->get\_cpf() == cpf;} ) == true )

295 {

296 cout << "ERRO: CPF repetido" << endl;

297 return;

298 }

299

300 else if (cpf <= 0 || nome.empty() || (to\_string(cpf).length() != 11) ) //Verifica se o CPF informado eh valido

301 {

302 cout << "ERRO: dados incorretos" << endl;

303 return;

304 }

305

306 else

307 {

308 Cliente\* novo\_cliente = new Cliente(cpf, nome);

309 \_Clientes.emplace\_back(novo\_cliente);

310 cout << "Cliente " << cpf << " cadastrado com sucesso" << endl;

311 }

312 }

#### void Locadora::cadastrar\_Filme (bool file, char tipo, int quantidade, int codigo, string titulo, string categoria)

Função responsável pelo cadastro de filmes.

190 {

191 //CONFERINDO DADOS

192 //OBS: NO MAIN, CONFERIR INPUT DE DADOS E IMPRIMIR ERROS ANTES DE CHAMAR A FUNÇÃO

193

194 if( \_Estoque.find( codigo ) != \_Estoque.end() )

195 {

196 cout << "ERRO: codigo repetido" << endl;

197 return;

198 }

199

200 switch(tipo){

201 case('D') : {

202 if( categoria == "" && file == false){

203 cout << "ERRO: dados incorretos" << endl;

204 return;

205 }

206 else{

207 Filme\* Novo\_Filme = new DVD(codigo, titulo, quantidade, categoria);

208 \_Estoque.emplace(codigo, Novo\_Filme);

209 }

210 }

211 case ('F') : {

212 if( categoria != "" && file == false){

213 cout << "ERRO: dados incorretos" << endl;

214 return;

215 }

216 else{

217 Filme\* Novo\_Filme = new Fita(codigo, titulo, quantidade);

218 \_Estoque.emplace(codigo, Novo\_Filme);

219 }

220 }

221 default : {

222 if(!file){

223 cout << "ERRO: dados incorretos" << endl;

224 return;

225 }

226 }

227 }

228

229 if( file == false ) {

230 cout << "Filme " << codigo << " cadastrado com sucesso" << endl;

231 }

232 }

#### void Locadora::devolver\_Filme (long cpf)

Função de devolução de filmes.

116 {

117 if ( this->buscar\_cliente( cpf ) == nullptr ) { return; };

118

119 vector <Filme\*> \*filmes\_alugados = &(this->buscar\_cliente( cpf )->\_Filmes\_Alugados);

120 float valor\_a\_pagar = 0.0;

121 float valor\_por\_filme = 0.0;

122 int dias = rand() % 14 + 1;

123

125 stringstream stream\_cliente;

126 stream\_cliente << "Cliente " << cpf << " "

127 << this->buscar\_cliente( cpf )->get\_nome()

128 << " devolveu os filmes:" << endl;

129

130 string linha\_cliente;

131 getline (stream\_cliente, linha\_cliente);

132 cout << linha\_cliente << endl;

133 \_Logs << linha\_cliente << endl;

134

135

136 for( auto filme\_devolvido : (\*filmes\_alugados) ){

137 filme\_devolvido->\_qtd\_disp += 1;

138 valor\_por\_filme = filme\_devolvido->calc\_valor\_locacao(dias);

139 valor\_a\_pagar += valor\_por\_filme;

140

142 stringstream stream\_filme;

143 stream\_filme << filme\_devolvido->get\_id()

144 << " R$" << valor\_por\_filme;

145

146 string linha\_filme;

147 getline(stream\_filme, linha\_filme);

148 cout << linha\_filme << endl;

149 \_Logs << linha\_filme << endl;

150

151 }

152 filmes\_alugados->clear();

153

154 cout << "Total a pagar: R$" << valor\_a\_pagar << endl;

155 \_Logs << "Total pago: R$" << valor\_a\_pagar << endl;

156 }

#### map< int, Filme \* > Locadora::getEstoque ()

Mapeamento de estoque/controle do Estoque.

159 { return \_Estoque; }

#### void Locadora::getNome ()

Obtém o nome da locadora.

#### void Locadora::imprimir\_clientes (char tipo\_ordenacao)

Retorna os clientes em ordem

331 {

332 vector <Cliente\*> clientes\_ordernados = \_Clientes;

333 if ( tipo\_ordenacao == 'C')

334 {

335 sort(clientes\_ordernados.begin(), clientes\_ordernados.end(), [](const Cliente\* c1, const Cliente\* c2){

336 return c1->get\_cpf () < c2->get\_cpf();

337 });

338 }

339

340 else if ( tipo\_ordenacao == 'N')

341 {

342 sort(clientes\_ordernados.begin(), clientes\_ordernados.end(), [](const Cliente\* c1, const Cliente\* c2){

343 return c1->get\_nome () < c2->get\_nome();

344 });

345 }

346

347 for ( auto cliente : clientes\_ordernados)

348 {

349 cout << cliente->get\_cpf() << " " << cliente->get\_nome() << endl;

350 };

351 }

#### void Locadora::imprimir\_Estoque (char tipo\_ordenacao)

Retorna os títulos que estão em estoque em ordem.

249 {

250 vector <Filme\*> filmes\_ordenados;

251 for( auto it : \_Estoque )

252 {

253 filmes\_ordenados.push\_back( it.second );

254 }

255

256 switch(tipo\_ordenacao){

257 case('T') : {

258 sort(

259 filmes\_ordenados.begin(),

260 filmes\_ordenados.end(),

261 [](const Filme\* F1,const Filme\* F2 ) { return F1->get\_titulo() <= F2->get\_titulo(); }

262 );

263 break;

264 }

265 case('C') : {

266 sort(

267 filmes\_ordenados.begin(),

268 filmes\_ordenados.end(),

269 [](const Filme\* F1,const Filme\* F2 ) { return F1->get\_id() <= F2->get\_id(); }

270 );

271 break;

272 }

273 default : {

274 cout << "ERRO: comando invalido" << endl;

275 return;

276 }

277 }

278

279 for( Filme\* it : filmes\_ordenados ) {

280 cout << it->get\_id() << " "

281 << it->get\_titulo() << " "

282 << it->get\_qtdDisp() << " "

283 << it->get\_tipo() << " "

284 << it->get\_categoria() << endl;

285 }

286

287

288 }

#### void Locadora::ler\_Arquivo\_de\_Estoque (string nome\_do\_arquivo)

Faz a leitura de um arquivo do estoque

162 {

163 ifstream arquivo\_de\_filmes;

164 arquivo\_de\_filmes.open( nome\_do\_arquivo );

165

166 char tipo;

167 int quantidade, codigo, contador;

168 string titulo, categoria, linha\_do\_arquivo;

169

170 int numero\_de\_filmes = 0;

171 if( arquivo\_de\_filmes.is\_open() )

172 {

173 while( getline( arquivo\_de\_filmes, linha\_do\_arquivo ) )

174 {

175 stringstream stream\_linha\_do\_arquivo;

176 stream\_linha\_do\_arquivo << linha\_do\_arquivo;

177 stream\_linha\_do\_arquivo >> tipo >> quantidade >> codigo >> titulo >> categoria;

178

179 this->cadastrar\_Filme(1, tipo, quantidade, codigo, titulo, categoria);

180 numero\_de\_filmes += 1;

181 }

182 arquivo\_de\_filmes.close();

183 cout << numero\_de\_filmes << " Filmes cadastrados com sucesso" << endl;

184 }

185 else

186 cout << "ERRO: arquivo inexistente" << endl;

187 }

#### void Locadora::remover\_cliente (long cpf)

Remove um cliente

314 {

315 auto it = find\_if(

316 \_Clientes.begin(),

317 \_Clientes.end(),

318 [cpf] (const Cliente\* cliente1){ return cliente1->get\_cpf() == cpf;}

319 );

320

321 if (it != \_Clientes.end())

322 {

323 \_Clientes.erase(it);

324 cout << "Cliente " << cpf << " removido com sucesso" << endl;

325 }

326 else{

327 cout << "ERRO: CPF inexistente" << endl;

328 };

329 }

#### void Locadora::remover\_Filme (int codigo)

Remove um filme

235 {

236 if( \_Estoque.find( codigo ) != \_Estoque.end() )

237 {

238 delete \_Estoque.find( codigo )->second;

239 \_Estoque.erase( codigo );

240 cout << "Filme " << codigo << " removido com sucesso" << endl;

241 }

242 else

243 {

244 cout << "ERRO: codigo inexistente" << endl;

245 }

246 }

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

TRABALHO FINAL/**Locadora.hpp**

TRABALHO FINAL/**Locadora.cpp**

# Arquivos

## Referência do Arquivo TRABALHO FINAL/cliente.cpp

#include "cliente.hpp"

#include <algorithm>

Referência do Arquivo TRABALHO FINAL/cliente.hpp

#include <iostream>

#include <string>

#include <vector>

#include "Filme.hpp"

### Componentes

class **Cliente**

## cliente.hpp

Ir para a documentação desse arquivo.

1

2 #ifndef CLIENTE\_H

3 #define CLIENTE\_H

4 #include <iostream>

5 #include <string>

6 #include <vector>

7

8 #include "Filme.hpp"

9

10 using namespace std;

11

12

13

14 class Cliente

15 {

16 friend class Locadora;

17 private:

18 string \_nome;

19 long \_cpf;

20 vector <Filme\*> \_Filmes\_Alugados;

21 public:

23 Cliente(long cpf, string nome);

25 ~Cliente();

26

27 string get\_nome() const;

28 long get\_cpf() const;

29

30 };

31

32

33 #endif

## Referência do Arquivo TRABALHO FINAL/dvd.cpp

#include "dvd.hpp"

## Referência do Arquivo TRABALHO FINAL/dvd.hpp

#include "Filme.hpp"

### Componentes

class **DVD.**

## dvd.hpp

Ir para a documentação desse arquivo.

1

2

3 #ifndef DVD\_H

4 #define DVD\_H

5 #include "Filme.hpp"

6

7 class DVD : public Filme {

8 private:

9 string \_categoria;

10

11 public:

13 DVD(int id, string titulo, int qtd\_disp, string categoria);

15 ~DVD();

17 virtual string get\_categoria();

19 virtual string get\_tipo() override;

21 float calc\_valor\_locacao (int dias) const override;

22

23

24

25 };

26

27 #endif

## Referência do Arquivo TRABALHO FINAL/Filme.cpp

#include "Filme.hpp"

## Referência do Arquivo TRABALHO FINAL/Filme.hpp

#include <iostream>

#include <string>

### Componentes

class **Filme**

## Filme.hpp

Ir para a documentação desse arquivo.

1

2

4

5 #ifndef FILME\_H

6 #define FILME\_H

7 #include <iostream>

8 #include <string>

9 using namespace std;

10

11

12 class Filme

13 {

14 friend class Locadora;

15 private:

16 int \_id;

17 string \_titulo;

18 int \_qtd\_disp;

19

20 public:

21 Filme(int id, string titulo, int qtd\_disp);

22 virtual ~Filme();

23

24 virtual string get\_categoria() = 0;

25 virtual string get\_tipo() = 0;

26 int get\_id() const;

27 string get\_titulo() const;

28 int get\_qtdDisp() const;

29

30 virtual float calc\_valor\_locacao (int dias) const = 0;

31 };

32

33 #endif

## Referência do Arquivo TRABALHO FINAL/fita.cpp

#include "fita.hpp"

#include <cstdlib>

## Referência do Arquivo TRABALHO FINAL/fita.hpp

#include "Filme.hpp"

### Componentes

class **Fita**.

## fita.hpp

Ir para a documentação desse arquivo.

1

2

3 #ifndef FITA\_H

4 #define FITA\_H

5 #include "Filme.hpp"

7 class Fita : public Filme {

8

9 private:

10 bool \_isRebobinada;

11

12 public:

14 Fita(int id, string titulo, int qtd\_disp);

16 ~Fita();

18 virtual string get\_categoria() override;

20 virtual string get\_tipo() override;

22 bool get\_isRebobinada() const;

24 float calc\_valor\_locacao (int dias) const override;

25

26 };

27

28

29

30 #endif

## Referência do Arquivo TRABALHO FINAL/Locadora.cpp

#include "Locadora.hpp"

#include <algorithm>

#include <fstream>

#include <cstdlib>

#include <sstream>

#include <map>

## Referência do Arquivo TRABALHO FINAL/Locadora.hpp

#include "Filme.hpp"

#include "dvd.hpp"

#include "fita.hpp"

#include "cliente.hpp"

#include <iostream>

#include <string>

#include <fstream>

#include <map>

#include <vector>

### Componentes

class **Locadora.**

## Locadora.hpp

Ir para a documentação desse arquivo.

1

2

3 #ifndef LOCADORA\_H

4 #define LOCADORA\_H

5

6 #include "Filme.hpp"

7 #include "dvd.hpp"

8 #include "fita.hpp"

9 #include "cliente.hpp"

10

11 #include <iostream>

12 #include <string>

13 #include <fstream>

14 #include <map>

15 #include <vector>

16 using namespace std;

17

19

20 class Locadora

21 {

22 private:

23 string \_Nome;

24 ofstream \_Logs;

25 map <int, Filme\*> \_Estoque;

26 vector <Cliente\*> \_Clientes;

27 public:

29 Locadora( string nome );

31 ~Locadora();

33 void getNome();

35 void alugar\_Filme( vector<int> codigos, long cpf );

36 void devolver\_Filme( long cpf );

38 void cadastrar\_cliente( long cpf, string nome);

39 void remover\_cliente(long cpf);

40 void imprimir\_clientes(char tipo\_ordenacao);

41 Cliente\* buscar\_cliente(long cpf);

43 map <int, Filme\*> getEstoque();

44

45 void ler\_Arquivo\_de\_Estoque( string nome\_do\_arquivo );

46 void cadastrar\_Filme( bool file, char tipo, int quantidade, int codigo, string titulo, string categoria ); //BOOL FILE DEVERÁ SER PREENCHIDO COMO 0 NO MAIN!!!

47 void remover\_Filme( int codigo );

48 void imprimir\_Estoque( char tipo\_ordenacao );

49

50 };

51

52

53 #endif

## Referência do Arquivo TRABALHO FINAL/main.cpp

#include <iostream>

#include <string>

#include "Filme.hpp"

#include "dvd.hpp"

#include "cliente.hpp"

#include "fita.hpp"

#include "Locadora.hpp"

#include <sstream>

#include <algorithm>

#include <ctype.h>

### Funções

#### bool isNotNumeric (const char & c)

Função auxiliar que verifica se um caractere não é numérico.

21 {

22 if ( isdigit(c) )

23 return false;

24 else

25 return true;

26 };

#### int main ()

Função principal que gerencia as operações da locadora.

28 {

29 string comando;

30 Locadora localiza("localiza");

31 while(cin>>comando){

32 if(comando=="LA"){

33 string nome\_arquivo;

34 cin >> nome\_arquivo;

35 localiza.ler\_Arquivo\_de\_Estoque( nome\_arquivo );

36 }

37 else if(comando=="CF"){

38 int codigo, quantidade;

39 char tipo;

40 string titulo, categoria;

41 cin >> tipo;

42 cin >> quantidade;

43 cin >> codigo;

44 cin >> titulo;

45 if(tipo == 'D')

46 cin >> categoria;

47 else

48 categoria = "";

49 localiza.cadastrar\_Filme(0, tipo, quantidade, codigo, titulo, categoria);

50 }

51 else if(comando=="RF"){

52 int codigo;

53 cin >> codigo;

54 localiza.remover\_Filme( codigo );

55 }

56 else if(comando=="LF"){

57 char letra;

58 cin >> letra;

59 localiza.imprimir\_Estoque( letra );

60 }

61 else if(comando=="CC"){

62 long cpf;

63 string nome;

64 cin>>cpf;

65 cin>>nome;

66 localiza.cadastrar\_cliente(cpf,nome);

67 }

68 else if(comando=="RC"){

69 long cpf;

70 cin>>cpf;

71 localiza.remover\_cliente(cpf);

72 }

73 else if(comando=="LC"){

74 char letra;

75 cin>>letra;

76 localiza.imprimir\_clientes(letra);

77 }

78 else if(comando=="AL"){

79 vector<int> codigos;

80 long cpf, novo\_codigo;

81 cin >> cpf;

82 string string\_codigo, linha\_input;

83

84 getline(cin, linha\_input);

85 stringstream istream;

86 istream << " " << linha\_input << " ";

87

88 while( getline(istream, string\_codigo, ' ') ){

89 if( any\_of( string\_codigo.begin(), string\_codigo.end(), isNotNumeric) ) {}

90 else if( string\_codigo.empty() == true ) {}

91 else {

92 stringstream stream\_codigos;

93 stream\_codigos << string\_codigo;

94 stream\_codigos >> novo\_codigo;

95 codigos.push\_back( novo\_codigo );

96 }

97 }

98

99 localiza.alugar\_Filme(codigos, cpf);

100 }

101 else if(comando=="DV"){

102 long cpf;

103 cin>>cpf;

104 localiza.devolver\_Filme(cpf);

105 }

106 else if(comando=="FS"){

107 break;

108 }

109 else

110 cout << "ERRO: comando inexistente" << endl;

111

112 }

113

114

115 return 0;

116 }