

**UNIÃO EDUCACIONAL DO NORTE – UNINORTE**

**MANASSÉS DE OLIVEIRA CARVALHO**

**MYLENA FRANKLIM DE PAIVA**

**VALDENILSON DE MELO NASCIMENTO**

**WAYRA AYANNE ALEXANDRE ZEVALLOS**

**SISTEMA DE BIBLIOTECA – DOCUMENTAÇÃO**

**RIO BRANCO – AC**

**2019**

**UNIÃO EDUCACIONAL DO NORTE – UNINORTE**

**MANASSÉS DE OLIVEIRA CARVALHO**

**MYLENA FRANKLIM DE PAIVA**

**VALDENILSON DE MELO NASCIMENTO**

**WAYRA AYANNE ALEXANDRE ZEVALLOS**

**SISTEMA DE BIBLIOTECA – DOCUMENTAÇÃO**

Trabalho apresentado ao Curso de Sistema de Informação da União Educacional do Norte – UNINORTE como requisito para nota parcial da disciplina de P.O.O (Programação Orientada a Objetos) na N2, ministrada pelo professor Edkallen Lima.

**RIO BRANCO – AC**

**2019**

## **1. ESCOPO**

O sistema desenvolvido tem por objetivo facilitar a gestão de funcionários em uma biblioteca, fornecendo permissão para cadastro dos mesmos, por meio de um administrador, e acesso a cadastramento de clientes por funcionários específicos da biblioteca. Assim como também permite o cadastramento e busca de livros e de suas devidas quantidade em estoque. Permite, também, o processo de empréstimos de livros por meio de um ambiente facilitado, tanto para o funcionário que realizará a operação, quanto para o cliente, elaborando um relatório da operação para ambos. Caso haja diária excedida no empréstimo pelo cliente, o sistema também realizar um o cálculo de multa no momento da devolução, constando todas as informações, da locação a devolução, em um relatório final para o cliente e funcionário. O sistema também permite a elaboração de relatórios por tipo de livro, constando suas informações necessárias.

### **1.1 FINALIDADE**

Este trabalho trata do sistema simplificado de gestão de uma biblioteca, que executará o controle sobre as operações feitas com os livros, e também sobre o fluxo de informações geradas a partir dessas operações, facilitando o trabalho dos funcionários por meio de um sistema de fácil entendimento, para que se possa manter as informações organizadas.

## **1.2 OBJETIVOS**

Nesta seção serão apresentados o objetivo geral e os específicos.

### **1.2.1 OBJETIVO GERAL**

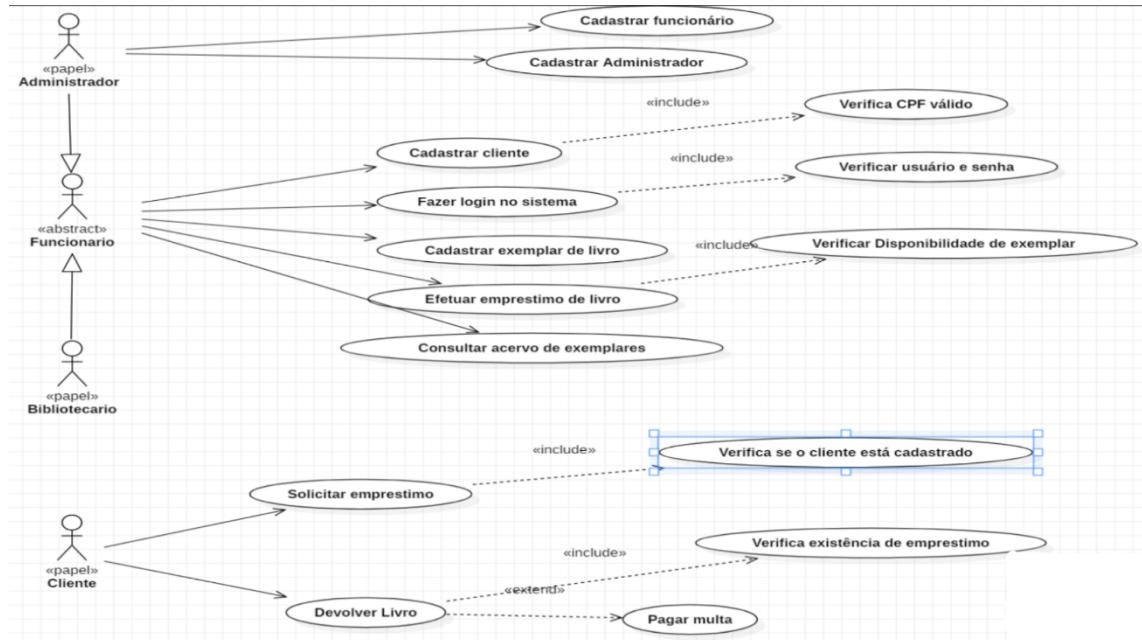
O objetivo geral deste trabalho é desenvolver um sistema de biblioteca que permita realizar de forma prática o controle e a gestão de informações geradas pela mesma, tendo como retorno relatórios das principais operações.

### **1.2.2 OBJETIVOS ESPECÍFICOS**

A seguir são listados os objetivos específicos deste trabalho, que visam em conjunto atingir o objetivo principal.

- Facilitar a organização de informações quanto ao acervo da biblioteca;
- Exemplificar o processo de empréstimo e devolução de livros;
- Exemplificar o processo de cadastramento de funcionários e clientes;
- Possibilitar o entendimento quanto ao funcionamento de um sistema bibliotecário.

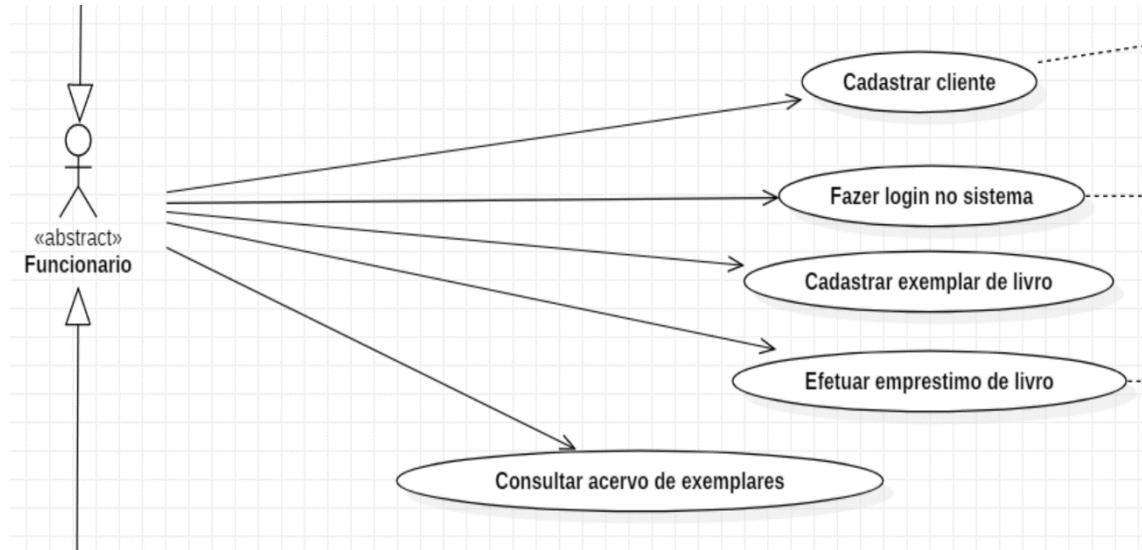
### **1.3 DESCRIÇÃO DOS CASOS DE USO**



\*Lista de atores do sistema de biblioteca

Administrador	Funcionário	Bibliotecário	Cliente
---------------	-------------	---------------	---------

- **Funcionário**



A classe funcionário é abstrata, ou seja, não pode ser instanciada e serve de modelo para outras classes, neste caso, as classes Administrador e Bibliotecário.

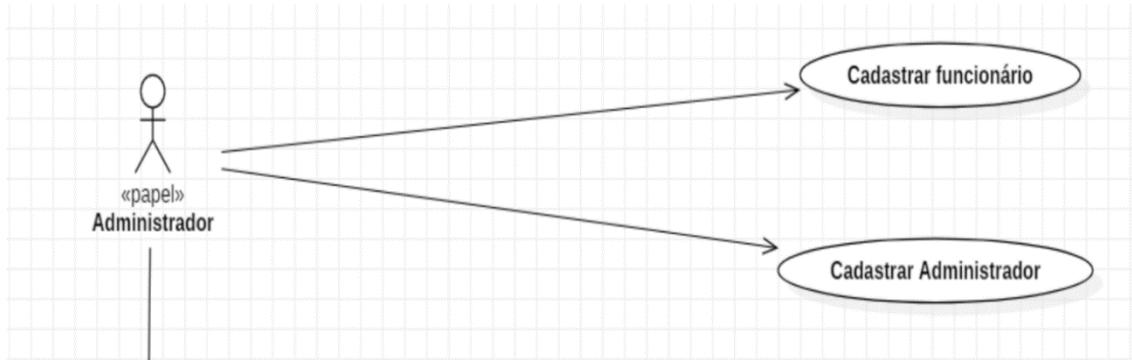
O funcionário, primeiramente, pode logar no sistema, isto é, ter acesso aos serviços de gerenciamento de empréstimos e devoluções de exemplares de livro. Para haver login é preciso que o funcionário seja cadastrado anteriormente pelo administrador da biblioteca e, claro, inserir os valores válidos no momento da entrada no sistema, pois o sistema faz uma verificação de validação de funcionário.

Um funcionário pode cadastrar novos livros no acervo, permitindo assim o acesso ao seu relatório. Ele é capaz de fazer empréstimos para os clientes cadastrados no sistema. Para isto, verifica-se a disponibilidade de exemplares no acervo.

Também é de acesso ao funcionário a consulta de exemplares disponíveis no acervo de livros da biblioteca.

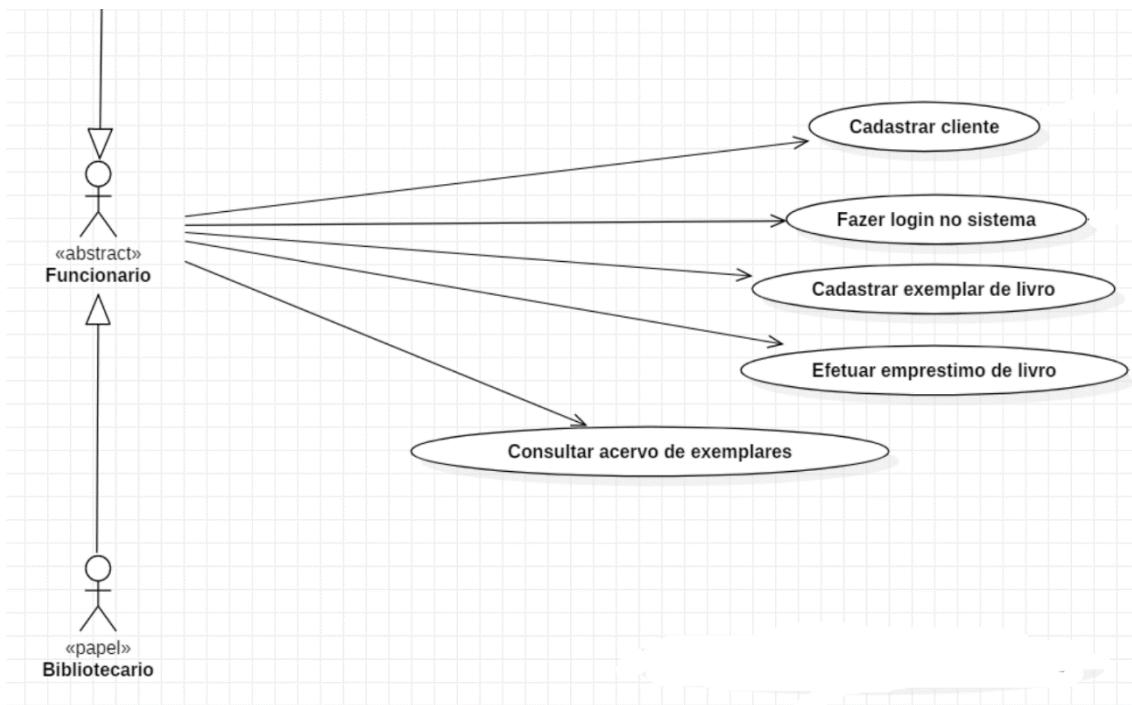
Um funcionário tem a possibilidade de cadastrar um novo cliente no sistema, mas, para isto acontecer, é necessário que o cliente informe um CPF válido, ou seja, precisa ter tamanho igual a 11 caracteres, e seguir com as regras de criação de CPF;

- **Administrador**



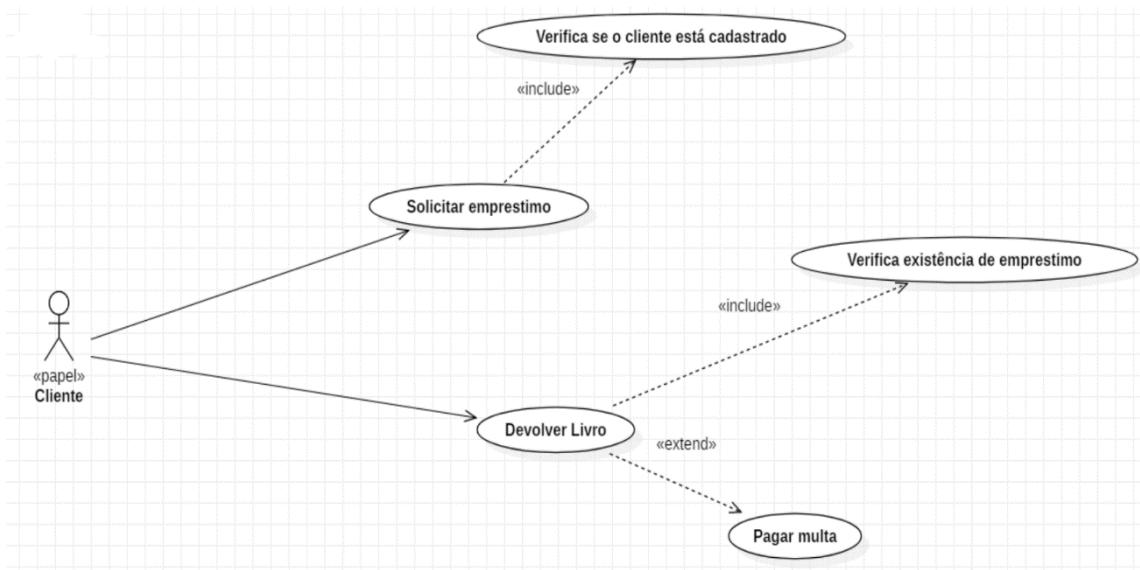
O administrador tem acesso a todas as funcionalidades do sistema, pois, além de herdar todas os métodos de funcionário, que é classe abstrata, também tem características próprias: cadastrar funcionários e cadastrar outros administradores.

- **Bibliotecário**



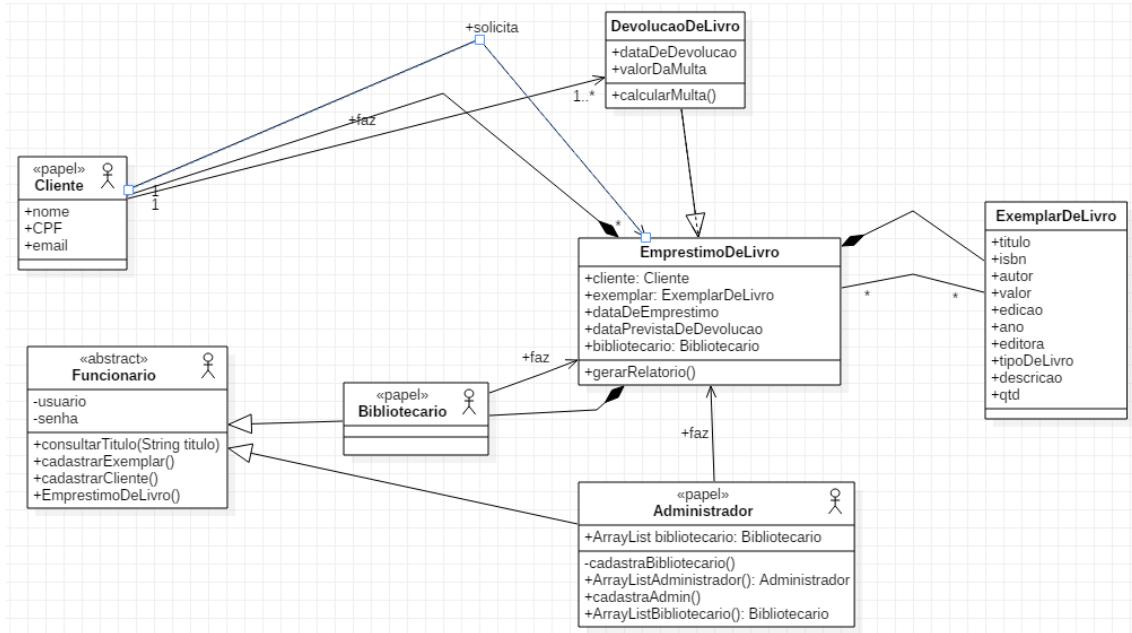
Este ator tem acesso a todas as funcionalidades da classe funcionário: cadastrar cliente, cadastrar exemplar de livro, efetuar empréstimo de livro e fazer a consulta deles no acervo.

- **Cliente**



O cliente, como já está cadastrado no sistema, pode solicitar alguns empréstimos de livros num determinado período de tempo e, caso este atrasse sua devolução, haverá como consequência uma multa.

## 1.4 DESCRIÇÃO DE CLASSES UML

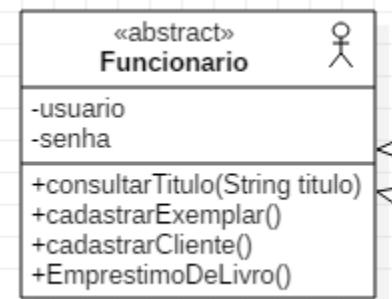


- **Cliente**



A classe Cliente serve para armazenar cada novo cadastro na biblioteca. Armazenando assim informações do usuário, tais como: nome, CPF e email.

- **Funcionario**



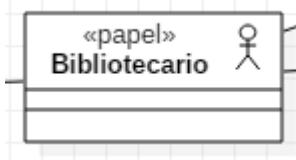
A classe Funcionario serve para cadastrar os funcionários da biblioteca. Armazena informações como: usuário e senha. Possui métodos para consultar título do livro, cadastrar exemplar, cadastrar cliente, e também tem acesso ao empréstimo de livros.

- **EmprestimoDeLivro**



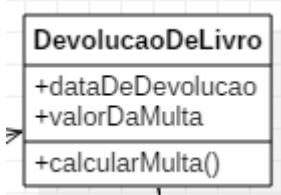
A classe EmprestimoDeLivros serve para armazenar cada empréstimo de livro realizado na biblioteca por clientes ou até mesmo funcionários. As informações armazenadas são: cliente que fez o empréstimo, exemplar do livro emprestado, data em que o empréstimo foi realizado e qual a data de devolução e qual bibliotecário concebeu o empréstimo ao cliente. Tem como método “gerarRelatorio” que gera um relatório contendo as informações preenchidas sobre o empréstimo.

- **Bibliotecario**



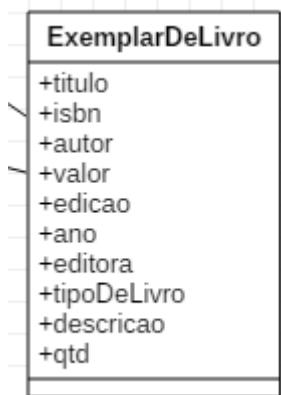
A classe Bibliotecario indica que é o responsável por realizar os empréstimos de livros para clientes da biblioteca. Está diretamente ligado ao Funcionario e ao EmprestimoDeLivros, tendo assim acesso as informações de cada um deles.

- **DevolucaoDeLivro**



Está classe serve para informar ao cliente o prazo para a devolução do livro de acordo com a data em que foi realizado o empréstimo. Gera também uma multa caso esse prazo não seja cumprido.

- **ExemplarDeLivro**



Está classe está reservada para guardar informações sobre exemplares de livros da biblioteca. Guarda dados como: título do livro, ISBN, autor, valor, edição, ano, editora, tipo de livro, descrição e quantidade.

- **Administrador**



A classe Administrador serve para identificar um funcionário um tanto quanto superior aos outros, pois, ela armazena uma lista que contém todos os bibliotecários. E possui métodos como: cadastrar novos bibliotecários, exibir uma lista dos administradores, cadastrar novos administradores, e exibir uma lista de todos os bibliotecários.

## 1.5 DESCRIÇÃO PORMENORIZADA DO SISTEMA

- Classes e suas Características:

**\*Observação:** todas as classes do sistema de biblioteca pertencem ao pacote 'PacoteSistema'.

- ADMIN

**\*Observação:** a classe 'Admin' é uma classe pública. A mesma herda todas as características

da classe 'Funcionario', tornando-se, então, uma especialização.

**Importações:** import java.util.ArrayList; import javax.swing.JOptionPane.

```
1 package PacoteSistema;
2
3 import java.util.ArrayList;
4 import javax.swing.JOptionPane;
5
```

Importa-se a classe ArrayList para armazenar eficientemente os objetos da classe Admin. Foram criados dois ArrayList com os seguintes nomes: 'bibliotecários' e 'admins'. O array de bibliotecários será responsável pelo armazenamento de todas as instâncias de 'Bibliotecário'. O array admins também tem o mesmo princípio, guardando os objetos de 'Admin' para posterior utilização.

A classe JOptionPane fornece recursos visuais para o sistema de biblioteca. Sem ela, as janelas de interação usuário/computador não seriam amigáveis.

**Construtores:** temos dois construtores para 'Admin': o primeiro tem como argumentos o usuário e a senha, ambos herdados pela classe mãe; o segundo não recebe nenhum argumento, ou seja, ele é o construtor padrão da nossa classe.

```
11
12     //Construtores de administrador
13     public Admin(String usuario, String senha) {
14         super(usuario, senha);
15     }
16     public Admin(){
17     }
```

## Métodos:

- **sevicosDisponiveis:** é um método sobrecarregado de 'Funcionário'. Este método tem retorno vazio, ou seja, void e tem características exclusivas por se tratar de acesso de um administrador. Esta exclusividade é mostrada na execução do programa, enquanto o bibliotecário tem acesso aos serviços de "CADASTRAR CLIENTE", "CADASTRAR LIVRO", "EMPRESTMO DE LIVRO", "CONSULTAR LIVRO", "DEVOLUCAO DE LIVRO", "CONSULTAR LIVRO POR TIPO", o administrador tem acesso a mais dois serviços: "CADASTRAR FUNCIONÁRIO" e "CADASTRAR ADMINISTRADOR";

```

19     //Lista os serviços disponíveis para o administrador
20     @Override
21     public void serviçosDisponiveis(){
22
23         Object[] itens = {"CADASTRAR CLIENTE", "CADASTRAR LIVRO", "EMPRESTIMO DE LIVRO", "CONSULTAR LIVRO", "CONSULTAR LIVRO POR TIPO", "CADASTRAR FUNCIONÁRIO", "DEVOLUÇÃO DE LIVRO", "CADASTRAR ADMINISTRADOR"};
24         Object selecionado = JOptionPane.showInputDialog(null, "ESCOLHA", "SERVIÇOS DISPONÍVEIS", JOptionPane.INFORMATION_MESSAGE, null, itens, itens[0]);
25
26         if(selecionado.equals("CADASTRAR CLIENTE")){
27             cadastrarCliente();
28         } else if (selecionado.equals("CADASTRAR LIVRO")){
29             cadastrarLivro();
30         } else if (selecionado.equals("EMPRESTIMO DE LIVRO")){
31             emprestimoDeLivro();
32         } else if (selecionado.equals("CADASTRAR FUNCIONÁRIO")){
33             cadastrarFuncionario();
34         } else if(selecionado.equals("DEVOLUÇÃO DE LIVRO")){
35             devolucaoDeLivro();
36         } else if(selecionado.equals("CADASTRAR ADMINISTRADOR")){
37             cadastrarAdmin();
38         } else if(selecionado.equals("CONSULTAR LIVRO POR TIPO")){
39             String tipoPesquisa = JOptionPane.showInputDialog(null, "\t\tVOCÊ ESCOLHEU A OPÇÃO CONSULTAR LIVROS POR TIPO\n\nPesquisar pelo tipo");
40             consultarTipo(tipoPesquisa);
41         } else {
42             String tituloPesquisa = JOptionPane.showInputDialog(null, "\t\tVOCÊ ESCOLHEU A OPÇÃO CONSULTAR LIVRO\n\nPesquisar pelo título");
43
44             if (consultarTitulo(tituloPesquisa)){
45
46                 comoProsseguir();
47
48             } else {
49                 JOptionPane.showMessageDialog(null, "Livro não encontrado!");
50
51                 comoProsseguir();
52             }
53         }
54     }
55 }
```

- **validaAdmin:** tem como argumentos o usuário e a senha dos administradores e retorna um boolean (true ou false), true se existe um administrador com os dados passados por parâmetro ou false caso não exista;

```

55
56     //Verifica no array a existência de administradores
57     public boolean validaAdmin(String usuario, String senha){
58         for (Admin admin : admins) {
59             if (usuario.equals(admin.usuario) && senha.equals(admin.senha)) {
60                 return true;
61             }
62         }
63         return false;
64     }
65 }
```

- **validaBibliotecario:** possui as mesmas características do validaAdmin, porém ele, logicamente, faz a busca do usuário e a senha no array 'bibliotecários';

```
65
66     //Verifica no array a existécia de bibliotecários
67     public boolean validaBibliotecario(String usuario, String senha){
68         for (Bibliotecario bibliotec : bibliotecarios) {
69             if (usuario.equals(bibliotec.usuario) && senha.equals(bibliotec.senha)) {
70                 return true;
71             }
72         }
73         return false;
74     }
75 }
```

- **inicializaFuncionarios:** é um método que não retorna nada. Ele cria um objeto do tipo 'Bibliotecário' chamado bruno e armazena-o em 'bibliotecários', após isso, cria outro objeto, mas este é do tipo administrador, que se chama Jon e o armazena em 'admins';

```
75
76
77     //Inicializa os funcionários de teste
78     public void inicializaFuncionarios(){
79
80         //Cria um bibliotecario e adiciona ao arraylist
81         Bibliotecario bruno = new Bibliotecario("Bruno", "123456");
82         bibliotecarios.add(bruno);
83
84         //Cria um administrador e adiciona ao arraylist
85         Admin jon = new Admin("Jon", "123456");
86         admins.add(jon);
87     }
88 }
```

- **cadastrarFuncionario:** sem argumentos e do tipo void, instancia um novo funcionário do sistema e o guarda em 'bibliotecarios';

```

87
88 //Cadastra novo funcionario no sistema
89 □ private void cadastrarFuncionario(){
90
91     //Atributos de funcionário
92     String usuarioF, senhaF;
93
94     usuarioF = JOptionPane.showInputDialog(null, "\t\tVOCÊ ESCOLHEU A OPÇÃO CADASTRO DE FUNCIONÁRIO\n\nDigite o usuario do funcionário");
95     senhaF = JOptionPane.showInputDialog(null, "Digite a senha");
96
97     Bibliotecario novoBibliotecario = new Bibliotecario(usuarioF, senhaF);
98     bibliotecarios.add(novoBibliotecario);
99
100    JOptionPane.showMessageDialog(null, "Funcionário cadastrado com sucesso!");
101
102    //0=Sim, 1=Não, 2=Cancelar
103    int novoCadastro = JOptionPane.showConfirmDialog(null, "Cadastrar novo funcionário?");
104    if(novoCadastro==0){
105        |   cadastrarFuncionario();
106    }
107
108    comoProsseguir();
109 }
110

```

- **cadastrarAdmin:** não tem argumentos como parâmetro e também não retorna nada, cria um novo objeto do tipo 'Admin' e salva este novo administrador em 'admins'.

```

110
111 //Cadastrar admin
112 □ private void cadastrarAdmin(){
113
114     String usuarioA, senhaA;
115
116     usuarioA = JOptionPane.showInputDialog(null, "\t\tVOCÊ ESCOLHEU A OPÇÃO CADASTRO DE ADMINISTRADOR\n\nDigite o usuario do administrador");
117     senhaA = JOptionPane.showInputDialog(null, "Digite a senha");
118
119     Admin novoAdmin = new Admin(usuarioA, senhaA);
120     admins.add(novoAdmin);
121
122     JOptionPane.showMessageDialog(null, "Administrador cadastrado com sucesso!");
123
124     //0=Sim, 1=Não, 2=Cancelar
125     int novoCadastro = JOptionPane.showConfirmDialog(null, "Cadastrar novo administrador?");
126     if(novoCadastro==0){
127         |   cadastrarAdmin();
128     }
129     comoProsseguir();
130 }
131
132

```

- **APLICATIVO BIBLIOTECA**

É classe pública e, também, principal (main).

**Importação:** import javax.swing.JOptionPane.

**Descrição:** criamos dois objetos, um da classe 'Admin' o outro da classe 'Bibliotecário'. O objeto da primeira classe anteriormente citada inicializa os funcionários padrão para a utilização do aplicativo, além disto ele também é responsável pela inicialização dos componentes da biblioteca (livros, clientes e empréstimos), a chamada do método 'servicosDisponiveis', a verificação da existência de administradores e bibliotecários no sistema. Foi criado um menu de duas opções de login, um de administrador e o outro de bibliotecário.

```
1 package FacoteSistema;
2
3 import javax.swing.JOptionPane;
4
5 public class AplicativoBiblioteca {
6
7     public static void main(String[] args) {
8
9         //Inicializa os funcionários do sistema
10        Admin administrador = new Admin();
11        administrador.inicializaFuncionarios();
12
13        //Intancia um bibliotecario usar os recursos de bibliotecário
14        Bibliotecario bibliotecario = new Bibliotecario();
15
16        Object[] itens = {"ADMINISTRADOR", "BIBLIOTECÁRIO"};
17        Object selecionado = JOptionPane.showInputDialog(null, "LOGAR COMO", "SISTEMA DE BIBLIOTECA", JOptionPane.INFORMATION_MESSAGE, null, itens, itens[0]);
18
19        if("ADMINISTRADOR".equals(selecionado)){
20
21            int novaTentativa;
22
23            do{
24                //Entra com o login do administrador
25                String usuarioTeste = JOptionPane.showInputDialog(null, "Admin, digite seu usuário");
26                String senhaTeste = JOptionPane.showInputDialog(null, "Agora sua senha");
27
28                if(administrador.validaAdmin(usuarioTeste, senhaTeste)){
29                    String adminLogado = administrador.NomeAdminLogado(usuarioTeste, senhaTeste);
30
31                    //Inicia o array de exemplares, clientes e emprestimos
32                    administrador.inicializaComponentes();
33
34                    JOptionPane.showMessageDialog(null, "Bem-vindo, " + usuarioTeste + ".");
35                    administrador.servicosDisponiveis();
36                } else {
37                    JOptionPane.showMessageDialog(null, "Administrador não encontrado");
38                }
39                // 0=Sim, 1=Não, 2=Cancelar
40                novaTentativa = JOptionPane.showConfirmDialog(null, "Tentar novamente?");
41
42            }while(novaTentativa==0);
43
44            JOptionPane.showMessageDialog(null, "Saindo...");
45            System.exit(0);
46        }
47    }
48}
```

```

47     } else if ("BIBLIOTECÁRIO".equals(selecionado)){
48
49         int novaTentativa;
50
51         do{
52             //Entra com o login do bibliotecário
53             String usuarioTeste = JOptionPane.showInputDialog(null, "Bibliotecário, digite seu usuário");
54             String senhaTeste = JOptionPane.showInputDialog(null, "Agora, digite sua senha");
55
56             if(administrador.validaBibliotecario(usuarioTeste, senhaTeste)){
57
58                 //Iniciaiza o array de exemplares, clientes e emprestimos
59                 bibliotecario.inicializaComponentes();
60
61                 JOptionPane.showMessageDialog(null, "Bem-vindo, " + usuarioTeste + ".");
62                 bibliotecario.servicosDisponiveis();
63             } else {
64                 JOptionPane.showMessageDialog(null, "Bibliotecário não encontrado");
65             }
66             // 0=Sim, 1=Não, 2=Cancelar
67             novaTentativa = JOptionPane.showConfirmDialog(null, "Tentar novamente?");
68
69         }while(novaTentativa==0);
70
71         JOptionPane.showMessageDialog(null, "Saindo...");
72         System.exit(0);
73     } else {
74         JOptionPane.showMessageDialog(null, "Saindo...");
75         System.exit(0);
76     }
77 }
78 }
```

- BIBLIOTECÁRIO

**Descrição:** Esta classe é pública e herda todas as características de 'Funcionário'. Tem dois construtores: um com a passagem de dois argumentos por parâmetro (usuário e senha), o outro vazio, isto é, padrão.

```

1 package PacoteSistema;
2
3 public class Bibliotecario extends Funcionario{
4
5     public Bibliotecario(String usuario, String senha) {
6         super(usuario, senha);
7     }
8
9     public Bibliotecario() {
10    }
11
12 }
13 }
```

- CLIENTE

**Descrição:** é classe pública e possui os seguintes atributos: nome, cpf e email. Ela tem um construtor com a passagem de três argumentos (nome, cpf, email).

```
1 package PacoteSistema;
2
3 public class Cliente {
4     public String nome;
5     public String CPF;
6     public String email;
7
8     public Cliente(String nome, String CPF, String email) {
9         this.nome = nome;
10        this.CPF = CPF;
11        this.email = email;
12    }
13
14
15 }
16
```

- DEVOLUCAO DE LIVRO

**Descrição:** é uma classe pública. Tem alguns atributos, como: data de devolução, valor da multa e total da multa. Contém, também, um construtor com passagem de todos os atributos ditos anteriormente.

```
1 package PacoteSistema;
2
3 public class DevolucaoDeLivro {
4     Integer []dataDevolucao;
5     float valorMulta;
6     float totalMulta;
7
8     public DevolucaoDeLivro(Integer[] dataDevolucao, float valorMulta, float totalMulta) {
9         this.dataDevolucao = dataDevolucao;
10        this.valorMulta = valorMulta;
11        this.totalMulta = totalMulta;
12    }
13
14 }
```

- EMPRESTIMO DE LIVRO

**Descrição:** é uma classe pública, possuinte dos seguintes atributos: nome do cliente, cpf do

cliente, exemplar de livro, funcionário responsável e a data de empréstimo realizado. Ela é composta por um construtor com passagem de parâmetros de todos os atributos anteriormente citados.

```
1 package PacoteSistema;
2
3 public class EmprestimoDeLivro {
4     public String nomeCliente, dataPrevistaDevolucao, cpfCliente, exemplar, funcionario;
5     Integer[] dataDeEmprestimo;
6
7     public EmprestimoDeLivro() {
8     }
9
10    public EmprestimoDeLivro(String nomeCliente, String dataPrevistaDevolucao, String cpfCliente, String exemplar, String funcionario, Integer[] dataDeEmprestimo) {
11        this.nomeCliente = nomeCliente;
12        this.dataPrevistaDevolucao = dataPrevistaDevolucao;
13        this.cpfCliente = cpfCliente;
14        this.exemplar = exemplar;
15        this.funcionario = funcionario;
16        this.dataDeEmprestimo = dataDeEmprestimo;
17    }
18
19
20}
21
```

## ○ EXEMPLAR

**Descrição:** é uma classe pública com vários atributos, os quais são: título, isbn, autor, valor do livro, edição, ano, editora, tipo de livro, descrição e quantidade. Contém método construtor com todos os atributos como argumento, e um outro método construtor padrão.

```

1 package PacoteSistema;
2
3 public class ExemplarL {
4
5     public String titulo;
6     public String isbn;
7     public String autor;
8     public String valor;
9     public String edicao;
10    public String ano;
11    public String editora;
12    public String tipoDeLivro;
13    public String descricao;
14    public int qtd;
15
16    public ExemplarL(String titulo, String isbn, String autor, String valor, String edicao, String ano, String editora, String tipoDeLivro, String descricao, int qtd) {
17        this.titulo = titulo;
18        this.isbn = isbn;
19        this.autor = autor;
20        this.valor = valor;
21        this.edicao = edicao;
22        this.ano = ano;
23        this.editora = editora;
24        this.tipoDeLivro = tipoDeLivro;
25        this.descricao = descricao;
26        this.qtd = qtd;
27    }
28
29    public ExemplarL() {
30    }
31
32}

```

## o FUNCIONÁRIO

**Descrição:** classe pública e abstrata, isto é, não pode ser instanciada. Serve de modelo para as classes 'Administrador' e 'Bibliotecário'

**Importações:** import static PacoteSistema.ValidaCPF.isCPF; import java.util.ArrayList; import java.util.Arrays; import java.util.Calendar e import javax.swing.JOptionPane.

```

1 package PacoteSistema;
2
3 import static PacoteSistema.ValidaCPF.isCPF;
4 import java.util.ArrayList;
5 import java.util.Arrays;
6 import java.util.Calendar;
7 import javax.swing.JOptionPane;
8

```

As importações import static PacoteSistema.ValidaCPF.isCPF, import java.util.Arrays e import java.util.Calendar são fundamentais para o funcionamento serviços do sistema. Valida CPF verifica se o cpf do cliente que desejamos cadastrar está correto.

Os Arrays auxiliam na exibição de dados de um array dentro de um 'JOptionPane.showMessageDialog(...)'; e Calendar retorna um vetor de Interger de 3 posições: a [0] representa o dia, [1] o mês e [2] o ano.

Ela possui dois atributos: o usuário e a senha de acesso do funcionário no sistema.

O construtor tem como parâmetro os atributos de funcionário.

### ArrayList:

- ArrayList<Cliente> clientes;
- ArrayList<ExemplarL> exemplares;
- ArrayList<EmprestimoDeLivro> emprestimo;
- ArrayList<DevolucaoDeLivro> devolucoes.

```
① public abstract class Funcionario {  
10  
11     //Atributos de funcionário  
12     public String usuario;  
13     public String senha;  
14  
15     //Arrays  
16     ArrayList<Cliente> clientes = new ArrayList();  
17     ArrayList<ExemplarL> exemplares = new ArrayList();  
18     ArrayList<EmprestimoDeLivro> emprestimos = new ArrayList();  
19     ArrayList<DevolucaoDeLivro> devolucoes = new ArrayList();  
20  
21     //Construtores para outras classes especializadas de funcionário  
22     public Funcionario(String usuario, String senha) {  
23         this.usuario = usuario;  
24         this.senha = senha;  
25     }  
26 }
```

### Métodos:

- Serviços disponíveis

**Descrição:** método void sem passagem por parâmetro. Ele fornece todos os serviços de acesso ao sistema pelo bibliotecário, entre estes serviços estão: "CADASTRAR CLIENTE", "CADASTRAR LIVRO", "EMPRESTMO DE LIVRO", "CONSULTAR LIVRO", "DEVOLUCAO DE LIVRO" e "CONSULTAR LIVRO POR TIPO".

```

30     //Lista os serviços disponíveis para o funcionário
31     public void servicosDisponiveis(){
32
33         //Menu de serviços
34         Object[] itens = {"CADASTRAR CLIENTE", "CADASTRAR LIVRO", "EMPRESTMO DE LIVRO", "CONSULTAR LIVRO", "DEVOLUCAO DE LIVRO", "CONSULTAR LIVRO POR TIPO"};
35         Object selecionado = JOptionPane.showInputDialog(null,"ESCOLHA", "SERVIOS DISPONIVEIS", JOptionPane.INFORMATION_MESSAGE, null, itens, itens[0]);
36
37         if(selecionado.equals("CADASTRAR CLIENTE")){
38             cadastrarCliente();
39         } else if (selecionado.equals("CADASTRAR LIVRO")){
40             cadastrarLivro();
41         } else if (selecionado.equals("EMPRESTMO DE LIVRO")){
42             emprestimoDeLivro();
43         } else if(selecionado.equals("DEVOLUCAO DE LIVRO")){
44             devolucaoDeLivro();
45         } else if(selecionado.equals("CONSULTAR LIVRO POR TIPO")){
46             String tipoPesquisa = JOptionPane.showInputDialog(null, "\t\tVOCÊ ESCOLHEU A OPÇÃO CONSULTAR LIVROS POR TIPO\n\nPesquisar pelo tipo");
47             consultarTipo(tipoPesquisa);
48         }else {
49             String tituloPesquisa = JOptionPane.showInputDialog(null, "\t\tVOCÊ ESCOLHEU A OPÇÃO CONSULTAR LIVRO\n\nPesquisar pelo titulo");
50
51             if (consultarTitulo(tituloPesquisa)){
52
53                 } else {
54                     JOptionPane.showMessageDialog(null, "Livro não encontrado!");
55                 }
56                 comoProsseguir();
57             }
58         }

```

➤ Cadastro de cliente:

**Descrição:** método void sem passagem por parâmetro. Após o preenchimento de todos os dados do cliente, este método instancia um novo cliente e adiciona ao array 'clientes'.

```

60 //Intancia um novo cliente no array 'clientes'
61 public void cadastrarCliente(){
62
63     //Atributos de cliente
64     String nomeC, CPFC, emailC;
65
66     nomeC = JOptionPane.showInputDialog(null, "\t\tVOCÊ ESCOLHEU A OPÇÃO CADASTRO DE CLIENTE\n\nDigite o nome");
67     String CPFTeste = JOptionPane.showInputDialog(null, "Agora digite o CPF");
68
69     //Valida o CPF digitado
70     do{
71         if (isCPF(CPFTeste)==false){
72             CPFTeste = JOptionPane.showInputDialog(null, "Digite um CPF válido!");
73         }
74     }while(isCPF(CPFTeste)==false);
75
76     //Recebe o CPF válido
77     CPFC = CPFTeste;
78
79     emailC = JOptionPane.showInputDialog(null, "Agora digite seu email");
80
81     Cliente novoCliente = new Cliente(nomeC, CPFC, emailC);
82     clientes.add(novoCliente);
83
84     JOptionPane.showMessageDialog(null, "Cliente "+ nomeC + " cadastrado com sucesso!");
85
86     //0=Sim, 1=Não, 2=Cancelar
87     int novoCadastro = JOptionPane.showConfirmDialog(null, "Cadastrar novo cliente?");
88     if(novoCadastro==0){
89         cadastrarCliente();
90     }
91
92     comoProsseguir();
93 }

```

➤ Cadastrar livro:

**Descrição:** método void sem passagem por parâmetro. Quando finalizado todo o processo de inserção de dados, este método instancia um objeto do tipo 'ExemplarL' e o adiciona ao array 'exemplares'.

```

95     //Cadastra um novo livro no array 'exemplares'
96     public void cadastrarLivro()throws NumberFormatException {
97
98     //Atributos do livro
99     String tituloE, ISBN, autorE, edicaoE, anoE, editoraE, descricaoE, tipoDeLivro, valorE, qtdString;
100    int qtdInteiro = 0;
101
102    tituloE = JOptionPane.showInputDialog(null, "\t\tVOCÊ ESCOLHEU A OPÇÃO CADASTRAR LIVRO\n\nDigite o título");
103    ISBN = JOptionPane.showInputDialog(null, "Digite o ISBN (Ex.: 000-00-000-0000-0");
104    autorE = JOptionPane.showInputDialog(null, "Autor");
105    edicaoE = JOptionPane.showInputDialog(null, "Edição");
106    anoE = JOptionPane.showInputDialog(null, "Ano de publicação");
107    editoraE = JOptionPane.showInputDialog(null, "Editora responsável");
108    descricaoE = JOptionPane.showInputDialog(null, "Descrição");
109    tipoDeLivro = JOptionPane.showInputDialog(null, "Tipo de livro");
110    valorE = JOptionPane.showInputDialog(null, "Valor");
111
112    //Inserir um número inteiro a quantidade de livros disponíveis; utilizando o tratamento de exceção NumberFormatException
113    boolean continueLoop = true;
114    do{
115        try{
116            qtdString = JOptionPane.showInputDialog(null, "Quantidade");
117            qtdInteiro = Integer.parseInt(qtdString);
118            continueLoop = false;
119        }
120        catch(NumberFormatException entradaInvalida){
121            JOptionPane.showMessageDialog(null,"Entrada inválida! Exceção: " + entradaInvalida + ", Tente novamente: " );
122        }
123    }while(continueLoop);
124
125    //Instância um novo livro
126    ExemplarL novoLivro = new ExemplarL(tituloE, ISBN, autorE, valorE, edicaoE, anoE, editoraE, descricaoE, qtdInteiro);
127    exemplares.add(novoLivro);
128
129    JOptionPane.showMessageDialog(null, "Livro " + tituloE +" cadastrado com sucesso!");
130
131    // 0=Sim, 1=Não, 2=Cancelar
132    int novoCadastro = JOptionPane.showConfirmDialog(null, "Cadastrar novo livro?");
133    if(novoCadastro==0){
134        cadastrarLivro();
135    }
136
137    comoProssseguir();
138 }

```

➤ Consultar título:

**Descrição:** método booleano com passagem por parâmetro do tipo String, especificamente o título. Este método mostra todas as informações do livro encontrado e retorna true caso ache, senão, retorna false.

```

139
140     //Mostra os dados do livro e retorna true ou não o encontra e retorna false
141     public boolean consultarTitulo(String titulo){
142
143         for(int i=0;i<exemplares.size();i++){
144             if(exemplares.get(i).titulo.equals(titulo)){
145                 JOptionPane.showMessageDialog(null, "\t\tLIVRO ENCONTRADO!\n\n"
146                     + "Titulo ..... : " + exemplares.get(i).titulo
147                     + "\nAutor ..... : " + exemplares.get(i).autor
148                     + "\nISBN ..... : " + exemplares.get(i).isbn
149                     + "\nValor $ ..... : " + exemplares.get(i).valor
150                     + "\nEdição ..... : " + exemplares.get(i).edicao
151                     + "\nAno ..... : " + exemplares.get(i).ano
152                     + "\nEditora ..... : " + exemplares.get(i).editora
153                     + "\nTipo do livro ... : " + exemplares.get(i).tipoDeLivro
154                     + "\nDisponíveis ... : " + exemplares.get(i).qtd
155                     + "\n\nDescrição \n{ " + exemplares.get(i).descricao + " }");
156             }
157         }
158         return false;
159     }
160 }
161

```

➤ Consultar tipo de livro:

**Descrição:** método void com passagem por parâmetro do tipo String. Neste caso, a passagem é o tipo de livro a ser encontrado. Ao reunir todos os títulos do mesmo tipo num vetor, ele é apresentado numa única mensagem de saída todos os livros encontrados e, se não encontrar nenhum uma mensagem é exibida na tela.

```

161
162     //Mostra os dados do livro e retorna true ou não o encontra e retorna false
163     public void consultarTipo(String tipo){
164         ArrayList<String> livrosTipo = new ArrayList();
165
166         for(int i=0;i<exemplares.size();i++){
167             if(exemplares.get(i).tipoDeLivro.equalsIgnoreCase(tipo)){
168                 livrosTipo.add(exemplares.get(i).titulo);
169             }
170         }
171         if(livrosTipo.isEmpty()){
172             JOptionPane.showMessageDialog(null, "\t\tLIVROS NÃO ENCONTRADOS!\n\n");
173         } else {
174             JOptionPane.showMessageDialog(null, "LIVRO(S) ENCONTRADO(S)\n" + Arrays.toString(livrosTipo.toArray()));
175         }
176         comoProsseguir();
177     }
178

```

➤ Empréstimo de livro:

**Descrição:** método void sem passagem por parâmetro. Para haver um empréstimo de livro o cliente solicitante deve estar cadastrado no sistema da biblioteca. Se ele estiver, todo o processo de empréstimo será iniciado. Caso ele não esteja cadastrado, será perguntado se o funcionário deseja cadastrar o cliente.

```
179 //Realiza emprestimo
180 public void emprestimoDeLivro() {
181
182     //Atributos de emprestimo
183     String nomeClienteE, funcionarioE, dataPrevistaDevolucaoE, CPFClienteE;
184     Integer []dataEmprestimoE;
185     String exemplarEmprestimo = null;
186
187     String buscaCPF = JOptionPane.showInputDialog(null, "\t\tVOCÊ ESCOLHEU A OPÇÃO EMPRESTIMO DE LIVRO\n\nDigite o CPF do cliente");
188
189     //Verifica se o cliente está cadastrado no sistema e se sim faz o cadastro, senão, não o faz
190     if(consultarCliente(buscaCPF)){
191
192         //Atribui à variável criada o cliente cadastrado
193         CPFClienteE = buscaCPF;
194
195         //Atribui o nome do cliente por meio do método de consulta de clientes
196         nomeClienteE = consultarNomeCliente(buscaCPF);
197
198         imprimeCliente(CPFClienteE);
199
200         String adicionarLivro = JOptionPane.showInputDialog(null, "Nome do livro para emprestimo");
201
202         //verifica existência de livro e quantidade de exemplares
203         if(consultarTitulo(adicionarLivro)){
204             exemplarEmprestimo = adicionarLivro;
205             subtrairExemplar(adicionarLivro);
206
207         } else {
208             JOptionPane.showMessageDialog(null, "Livro não encontrado!");
209         }
210 }
```

```

206
207     } else {
208         JOptionPane.showMessageDialog(null, "Livro não encontrado!");
209     }
210
211     //Recebe a data atual de empréstimo
212     dataEmprestimoE = dataEmprestimo();
213
214     dataPrevistaDevolucaoE = JOptionPane.showInputDialog(null, "Data prevista de devolução");
215     funcionarioE = JOptionPane.showInputDialog(null, "Nome do funcionário responsável");
216
217     //Intancia um novo empréstimo
218     EmprestimoDeLivro newEmprestimo = new EmprestimoDeLivro(nomeClienteE, dataPrevistaDevolucaoE,
219                 CPFClienteE, exemplarEmprestimo, funcionarioE, dataEmprestimoE);
220
221     emprestimos.add(newEmprestimo);
222
223     JOptionPane.showMessageDialog(null, "Empréstimo realizado com sucesso!");
224
225     imprimeRelatorioE(newEmprestimo);
226
227     // 0=Sim, 1=Não, 2=Cancelar
228     int novoEmprestimo = JOptionPane.showConfirmDialog(null, "Realizar novo empréstimo?");
229     if(novoEmprestimo==0){
230         cadastrarLivro();
231     }
232
233     } else {
234         JOptionPane.showMessageDialog(null, "Cliente não encontrado!");
235         //0=Sim, 1=Não, 2=Cancelar
236         int novoCadastro = JOptionPane.showConfirmDialog(null, "Cadastrar novo cliente?");
237         if(novoCadastro==0){
238             cadastrarCliente();
239         }
240     }
241
242     comoProsseguir();
243 }

```

➤ Data de empréstimo:

**Descrição:** método com retorno de vetor de integer sem passagem por parâmetro. Aqui é feito uma instanciação da classe Calendar e, posteriormente, armazenado seus valores de dia, mês e ano no vetor de retorno.

```

245     //Retorna data atual
246     public Integer[] dataEmprestimo() {
247
248         Integer []dataEmprestimo = new Integer[3];
249
250         Calendar hoje = Calendar.getInstance();
251
252         int dia = hoje.get(Calendar.DAY_OF_MONTH);
253         int mes = hoje.get(Calendar.MONTH);
254         int ano = hoje.get(Calendar.YEAR);
255
256         dataEmprestimo[0] = dia;
257         dataEmprestimo[1] = mes + 1;
258         dataEmprestimo[2] = ano;
259
260     }

```

- Consultar cliente:

**Descrição:** método booleano com passagem por parâmetro do tipo String. Ele percorre todo o arraylist de 'clientes' e se encontrar o cpf passado como parâmetro em sua chamada, retorna true, senão, retorna false;

```
262      //Retorna cliente cadastrado ou não
263      public boolean consultarCliente(String cpf){
264          for (Cliente cliente : clientes) {
265              if (cliente.CPF.equals(cpf)) {
266                  return true;
267              }
268          }
269          return false;
270      }
```

- Consultar nome do cliente:

**Descrição:** método com retorno de uma String e passagem por parâmetro do mesmo tipo. Ele é usado para capturar o nome do cliente na hora do empréstimo.

```
272      //Retorna o nome do cliente cadastrado
273      public String consultarNomeCliente(String cpf){
274          for (Cliente cliente : clientes) {
275              if (cliente.CPF.equals(cpf)) {
276                  return cliente.nome;
277              }
278          }
279          return "";
280      }
```

- Imprime dados do cliente:

**Descrição:** método void com passagem de parâmetro do tipo String (CPF). Ao encontrar o cliente no array 'clientes' é mostrado todas as informações do cliente.

```
282 //Imprime os dados do cliente
283 public void imprimeCliente(String cpf){
284     for (Cliente cliente : clientes) {
285         if (cliente.CPF.equals(cpf)) {
286             JOptionPane.showMessageDialog(null, "DADOS DO CLIENTE CADASTRADO\n\nCliente: " + cliente.nome + "\nCPF: " + cliente.CPF + "\nEmail: " + cliente.email);
287         }
288     }
289 }
```

- Subtrai exemplar:

**Descrição:** método void com passagem de parâmetro do tipo String (titulo). Esta funcionalidade retira uma unidade do acervo de livros.

```
291 //Subtrai a quantidade de exemplares ao acervo da biblioteca
292 public void subtrairExemplar(String titulo){
293     for (ExemplarL exemplare : exemplares) {
294         if (titulo.equals(exemplare.titulo)) {
295             exemplare.qtd = exemplare.qtd - 1;
296         }
297     }
298 }
```

- Imprime relatório de empréstimo:

**Descrição:** método void com passagem de parâmetro do tipo 'EmprestimoDeLivro'. Todas as informações que são de empréstimo serão mostradas na tela.

```
300 //Imprime o relatorio de emprestimo
301 public void imprimeRelatorioE(EmprestimoDeLivro emprestimo){
302
303     JOptionPane.showMessageDialog(null, "RELATÓRIO DE EMPRESTIMO\n\nCliente: " + emprestimo.nomeCliente + "\nCPF do cliente: "
304     + emprestimo.cpfCliente + "\nData de emprestimo: dia " + emprestimo.dataDeEmprestimo[0] + "/" + emprestimo.dataDeEmprestimo[1] + "/"
305     + emprestimo.dataDeEmprestimo[2] + "\nPossivel data de devolução: " + emprestimo.dataPrevistaDevolucao + "\nFuncionário responsável: "
306     + emprestimo.funcionario + "\nExemplar: " + emprestimo.exemplar);
307 }
```

- Inicializa componentes do sistema:

**Descrição:** método void sem passagem por parâmetro. Todos os componentes necessários para a funcionalidade do sistema são instanciados.

```

309 //Inicializa os componentes do funcionário
310 public void inicializaComponentes(){
311
312     //Instância(s) de livro
313     ExemplarL estruturas = new ExemplarL("Estruturas de dados e algoritmos em C++", " 978-85-221-2665-1", "Adam Drozdek", "90.00", "24", "2016",
314     "Cengage Learning", "CC", "O livro enfatiza especialmente a conexão entre a estrutura de dados e seus algoritmos, "
315     + "incluindo uma análise da complexidade dos algoritmos", 5);
316     exemplares.add(estruturas);
317
318     ExemplarL uml = new ExemplarL("UML Essencial", " 0-321-19368-7", "Martin Fowler", "124.9", "34", "2005",
319     "Bookman", "CC", "UML (Unified Modeling Language) é uma família de notações gráficas, apoiada por "
320     + "um metamodelo único, que ajuda na descrição e no projeto de sistemas de software, particularmente daqueles construídos utilizando o estilo orientado a objetos (OO)", 2);
321     exemplares.add(uml);
322
323     ExemplarL cplusplus = new ExemplarL("Programação em C++", " 978-85-8055-026-9", "Luis Joyanes Aguilar", "64.52", "24", "2008",
324     "Mc Graw Hill", "CC", "Linguagens de programação - C++", "Este livro foi feito para ensinar a programar utilizando C++ e não para "
325     + "ensinar C++, ainda que também pretenda atingir esse objetivo", 3);
326     exemplares.add(cplusplus);
327
328
329     //Instância(s) de cliente
330     Cliente luiz = new Cliente("Luiz", "29828420090", "luiz@gmail.com");
331     clientes.add(luiz);
332     Cliente lucas = new Cliente("Lucas", "65761848082", "lucas@gmail.com");
333     clientes.add(lucas);
334     Cliente jose = new Cliente("José", "03205564251", "jose@gmail.com");
335     clientes.add(jose);
336
337
338     //Instâncias de empréstimos
339     Integer []dataEmprestimoLuiz = new Integer[3];
340     dataEmprestimoLuiz[0] = 21;
341     dataEmprestimoLuiz[1] = 6;
342     dataEmprestimoLuiz[2] = 2019;
343     EmprestimoDeLivro emprestimoLuiz = new EmprestimoDeLivro("Luiz", "25/6/2019", "29828420090", "Programação em C++", "Jon", dataEmprestimoLuiz);
344     emprestimos.add(emprestimoLuiz);
345
346 }
```

- Como prosseguir:

**Descrição:** método void sem passagem por parâmetro. Nele estão as opções de voltar à serviços disponíveis ou sair do sistema.

```

348 //Opções de continuação
349 public void comoProsseguir(){
350
351     Object[] opcoes = {"VOLTAR À SERVIÇOS DISPONÍVEIS", "FINALIZAR"};
352     Object escolhido = JOptionPane.showInputDialog(null, "SELEÇÃO", "O QUE DESEJA FAZER?", JOptionPane.INFORMATION_MESSAGE, null, opcoes, opcoes[0]);
353
354     if ("VOLTAR À SERVIÇOS DISPONÍVEIS".equals(escolhido)){
355         servicosDisponiveis();
356     } else {
357         JOptionPane.showMessageDialog(null, "Saindo...");
358         System.exit(0);
359     }
360 }
```

- Existe empréstimo:

**Descrição:** método booleano com passagem por parâmetro do tipo String(CPF). Retorna verdadeiro caso haja empréstimo para este cpf, retorna falso se não encontrar.

```

361
362     //Verificar a existência de emprestimo através do CPF do cliente
363     public boolean existeEmprestimo(String CPF){
364         for(EmprestimoDeLivro empres : emprestimos){
365             if(empres.cpfCliente.equals(CPF)){
366                 return true;
367             }
368         }
369         return false;
370     }
371

```

- Devolução de livro:

**Descrição:** método void sem passagem por parâmetro. Para a devolução do livro é necessário que o cliente informe o cpf para a verificação do empréstimo no sistema.

```

372     //Faz a instância de uma devolução
373     public void devolucaoDeLivro(){
374
375         //Atributos de devolução
376         Integer []dataDevolve;
377         final float multa = 3;
378         float totalMulta;
379
380         int diasExcedidos = 0;
381         String CPFCliente = JOptionPane.showInputDialog(null, "Digite o CPF do cliente para realizar a devolução");
382
383         //Verifica se a entrada é vazia
384         if(CPFCliente.isEmpty()){
385             do{
386                 CPFCliente = JOptionPane.showInputDialog(null, "Digite novamente o CPF");
387             }while(CPFCliente.isEmpty());
388         }
389
390         //Verifica a existência de emprestimo
391         if(existeEmprestimo(CPFCliente)){
392
393             //Como há a existência de emprestimo, recebe o índice do emprestimo
394             int indiceEmprestimo = indiceEmprestimo(CPFCliente);
395
396             JOptionPane.showMessageDialog(null, "Emprestimo efetuado dia " + emprestimos.get(indiceEmprestimo).dataDeEmprestimo[0] +
397             "/" + emprestimos.get(indiceEmprestimo).dataDeEmprestimo[1] + "/" + emprestimos.get(indiceEmprestimo).dataDeEmprestimo[2] +
398             "\nDia prevista de devolução: " + emprestimos.get(indiceEmprestimo).dataPrevistaDevolucao);
399
400             dataDevolve = dataDeDevolucao();
401
402             // 0=Sim, 1=Não, 2=Cancelar
403             int haAtraso = JOptionPane.showConfirmDialog(null, "Houve atraso na devolução do livro?");
404             if(haAtraso==0){
405                 JOptionPane.showMessageDialog(null, "Os dias de atraso são multiplicados por 3$!");
406             }
407         }
408     }

```

```

406
407     //Inserir um número inteiro
408     boolean continueLoop = true;
409     do{
410         try{
411             String diasE = JOptionPane.showInputDialog(null, "Quantos dias foram?");
412             diasExcedidos = Integer.parseInt(diasE);
413             continueLoop = false;
414         }
415         catch(NumberFormatException entradaInvalida){
416             JOptionPane.showMessageDialog(null,"Entrada inválida! Exceção: " + entradaInvalida + ", Tente novamente " );
417         }
418     }while(continueLoop);
419
420     totalMulta = calcularMulta(diasExcedidos, multa);
421
422     DevolucaoDeLivro novaDevolucao = new DevolucaoDeLivro(dataDevolve, multa, totalMulta);
423     devolucoes.add(novaDevolucao);
424
425     imprimeRelatorioComMulta(emprestimos.get(indiceEmprestimo), totalMulta, dataDevolve, diasExcedidos);
426
427 } else {
428     imprimeRelatorioE(emprestimos.get(indiceEmprestimo));
429 }
430
431     comoProsseguir();
432
433 } else {
434     JOptionPane.showMessageDialog(null,"Emprestimo inexistente!" );
435
436     comoProsseguir();
437 }
438
439

```

➤ Adicionar exemplar:

**Descrição:** método void com passagem de parâmetro do tipo String (titulo). Ele adiciona ao acervo da biblioteca uma unidade, neste caso, quando o cliente devolve o livro.

```

440
441     //Adiciona 1 a quantidade de exemplares na biblioteca
442     public void adicionarExemplar(String titulo){
443         for (ExemplarL exemplare : exemplares) {
444             if (titulo.equals(exemplare.titulo)) {
445                 exemplare.qtd = exemplare.qtd + 1;
446             }
447         }

```

➤ Índice do empréstimo:

**Descrição:** método com retorno int e passagem de parâmetro do tipo String (CPF).  
Retorna o índice do empréstimo para a exibição dos seus dados.

```
449     public int indiceEmprestimo(String CPF){  
450         for(int i=0;i<emprestimos.size();i++){  
451             if(emprestimos.get(i).cpfCliente.equals(CPF)){  
452                 return i;  
453             }  
454         }  
455         return 0;  
456     }
```

- Data de devolução:

**Descrição:** método com retorno de um vetor integer sem passagem por parâmetro. Retorna o vetor preenchido com a data atual

```
457     //Retorna data atual  
458     public Integer[] dataDeDevolucao(){  
459  
460         Integer []dataDevolucao = new Integer[3];  
461  
462         Calendar hoje = Calendar.getInstance();  
463  
464         int dia = hoje.get(Calendar.DAY_OF_MONTH);  
465         int mes = hoje.get(Calendar.MONTH);  
466         int ano = hoje.get(Calendar.YEAR);  
467  
468         dataDevolucao[0] = dia;  
469         dataDevolucao[1] = mes + 1;  
470         dataDevolucao[2] = ano;  
471     }  
472  
473 }
```

- Calcular multa:

**Descrição:** método com retorno float e passagem por parâmetro do tipo int e float. Retorna a multiplicação dos dias atrasados pela constante 3, que se refere a multa por um dia.

```
474     public float calcularMulta(int diasExcedidos, float multa){  
475         return diasExcedidos * multa;  
476     }
```

- Imprime relatório de devolução com multa

**Descrição:** método void com passagem de parâmetro do tipo 'Empréstimo', float, vetor de integer e int. Exibe na tela todos os dados da devolução com as devidas multas por atraso.

```
477  
478 //Imprime relatorio de devolucao com multa e a data definitiva de devolução  
479 public void imprimeRelatorioComMulta(EmprestimoDeLivro emprestimo, float multa, Integer []dataDevolucao, int atraso){  
480  
481 JOptionPane.showMessageDialog(null, "\tRELATÓRIO DE DEVOLUÇÃO\n\nCliente: " + emprestimo.nomeCliente + "\nCPF do cliente: "  
482 + emprestimo.cpfCliente + "\nData de emprestimo: dia " + emprestimo.dataDeEmprestimo[0] + "/" + emprestimo.dataDeEmprestimo[1]  
483 + "/" + emprestimo.dataDeEmprestimo[2] + "\nPossível data de devolução: " + emprestimo.dataPrevistaDevolucao + "\nFuncionário responsável: "  
484 + emprestimo.funcionario + "\nExemplar: " + emprestimo.exemplar + "\nData de devolução: dia " + dataDevolucao[0] + "/" + dataDevolucao[1] + "/"  
485 + dataDevolucao[2] + "\nAtraso de " + atraso + " dia(s)\nMulta #: " + multa);  
486 }  
487  
488 }
```

- VALIDA CPF

**Importação:** import java.util.InputMismatchException.

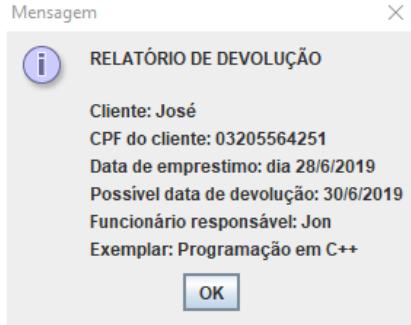
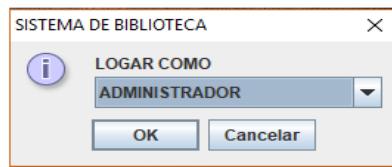
**Métodos:**

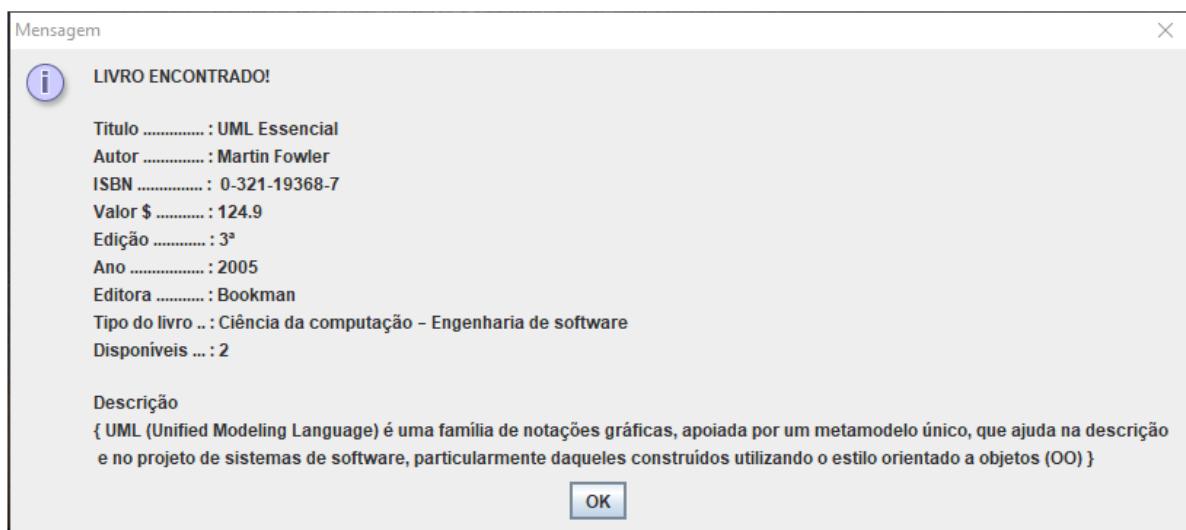
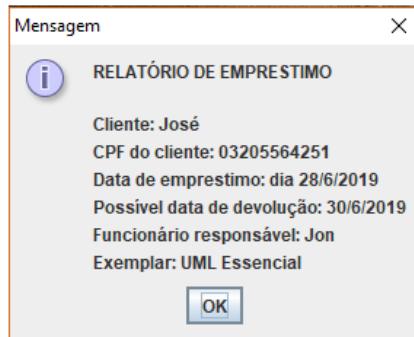
Esta classe possui o método estático booleano chamado isCPF, com um argumento chamado cpf. Ela recebe, processa e devolve ao solicitante um tipo lógico, true se o cpf é válido, false caso contrário.

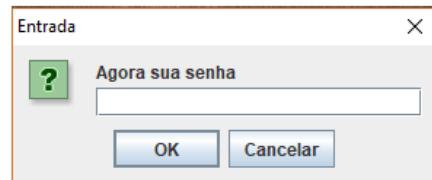
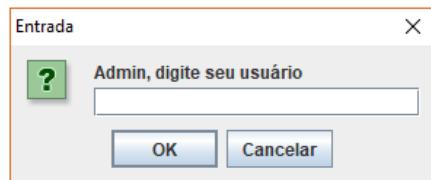
```
1 package PacoteSistema;
2
3
4 import java.util.InputMismatchException;
5
6 public class ValidaCPF {
7     public static boolean isCPF(String CPF) {
8         if (CPF.equals("00000000000") ||
9             CPF.equals("11111111111") ||
10            CPF.equals("22222222222") || CPF.equals("33333333333") ||
11            CPF.equals("44444444444") || CPF.equals("55555555555") ||
12            CPF.equals("66666666666") || CPF.equals("77777777777") ||
13            CPF.equals("88888888888") || CPF.equals("99999999999") ||
14            (CPF.length() != 11))
15             return(false);
16
17         char dig10, dig11;
18         int sm, i, r, num, peso;
19
20         try {
21
22             sm = 0;
23             peso = 10;
24             for (i=0; i<9; i++) {
25                 num = (int) (CPF.charAt(i) - 48);
26                 sm = sm + (num * peso);
27                 peso = peso - 1;
28             }
29
30             r = 11 - (sm % 11);
31             if ((r == 10) || (r == 11))
32                 dig10 = '0';
33             else dig10 = (char) (r + 48);
34
35             sm = 0;
36             peso = 11;
37             for(i=0; i<10; i++) {
38                 num = (int) (CPF.charAt(i) - 48);
39                 sm = sm + (num * peso);
40                 peso = peso - 1;
41             }
42
43             r = 11 - (sm % 11);
44             if ((r == 10) || (r == 11))
45                 dig11 = '0';
46             else dig11 = (char) (r + 48);
47
48             if ((dig10 == CPF.charAt(9)) && (dig11 == CPF.charAt(10)))
49                 return(true);
50             else return(false);
51             } catch (InputMismatchException erro) {
52                 return(false);
53             }
54         }
55
56     public static String imprimeCPF(String CPF) {
57         return(CPF.substring(0, 3) + "." + CPF.substring(3, 6) + "." +
58             CPF.substring(6, 9) + "-" + CPF.substring(9, 11));
59     }
60 }
61 }
```

## 1.6 TELAS DO SISTEMA EM USO

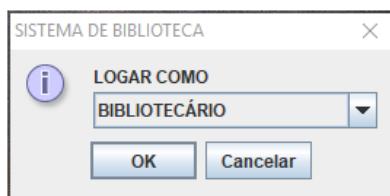
### 1.6.1 ADMIN

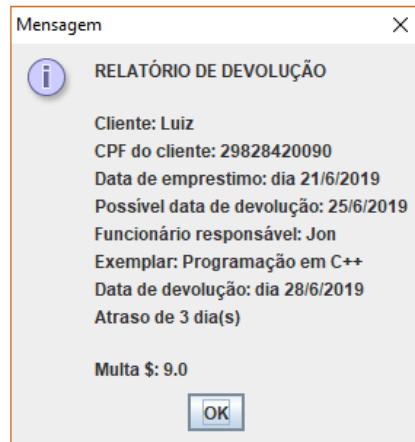


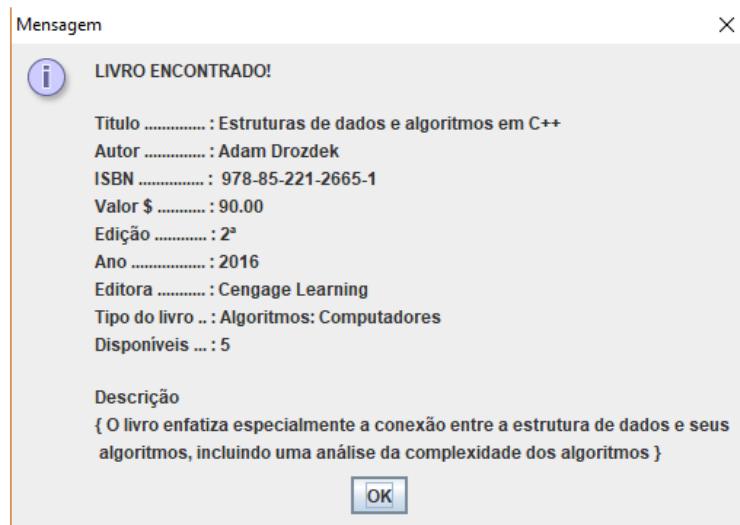




## 1.6.2 BIBLIOTECÁRIO







## BIBLIOGRAFIA

DEITEL, Paul Deitel, Harvey. JAVA: Como programar. 8 ed. São Paulo; Person Prentice Hall, 2010.

COLEMAN, Derek; Arnold Patrick. DESENVOLVIMENTO ORIENTADO A OBJETOS: O método fusion. Rio de Janeiro; Campus, 1996.

SANTOS, Rafael. INTRODUÇÃO À PROGRAMAÇÃO ORIENTADA A OBJETOS USANDO JAVA. Rio de Janeiro; Elsevier, 2013.

GUANABARA, Gustavo. CURSO DE POO JAVA: Programação orientada a objetos. 2016.

Disponível em: <http://www.youtube.com/watch?v=KILL63MeyMY&t=51s>. Acesso em: 15 jun 2019.