



Ministério da Educação  
**Universidade Tecnológica Federal do Paraná**  
Campus Apucarana

Coordenação de Engenharia de Computação- COENC  
Engenharia de Computação



---

## **RELATÓRIO TRABALHO FINAL- PROGRAMAÇÃO ORIENTADA A OBJETOS**

Giovanna Sanches Reghine.RA: 2271974  
João Pedro Cavani Meireles, RA: 2321424  
Michael Pariz Pereira, RA: 2321653

Apucarana  
12/12/2023

## **Sumário**

1. Introdução	3
2. Sobre o Projeto	4
3. Modelagem UML	5
4. Conceitos aplicados: Polimorfismo, Padrão de Projeto e Controle de erros.	16
5. Conclusão	18
6. Referências Bibliográficas	19

## **1. Introdução**

Este documento tem como propósito expor a implementação de uma central do assinante de uma empresa de Internet, desenvolvida em linguagem de programação Java, fundamentada nos princípios abordados durante as aulas de Programação Orientada a Objetos. Dentre esses princípios se destacam conceitos como Polimorfismo e Interface, entre outros. A consecução do projeto foi efetuada por meio da utilização da IDE NetBeans.

## **2. Sobre o Projeto**

Para a realização do projeto foi escolhido a central do assinante, pois se trata de uma interface muito útil na contemporaneidade, de modo a agilizar a vida cotidiana dos clientes das empresas pois os mesmos não perdem tempo em filas de atendimento. Podendo realizar para si próprio o autoatendimento por meio destas centrais.

O escopo do projeto abarca diversas funcionalidades típicas de uma central do assinante, incluindo a realização do cadastro de clientes, a contratação de planos, a emissão de segunda via, o relato de problemas de conexão por meio da interface do cliente, e, na perspectiva da interface do funcionário interno, a supervisão das solicitações dos clientes e a coordenação do envio de técnicos para solucionar as demandas apresentadas pelos clientes.

**Link para o Repositório com o Código Fonte:**

<https://github.com/JotaPeedro/ProjetoPoo>

### 3. Modelagem UML

Para a realização do projeto foram necessárias realizar os diagramas UML, tanto de casos de uso, classes, atividades etc.. , vistos em aula como suas respectivas tabelas.

#### Diagrama de casos de Uso

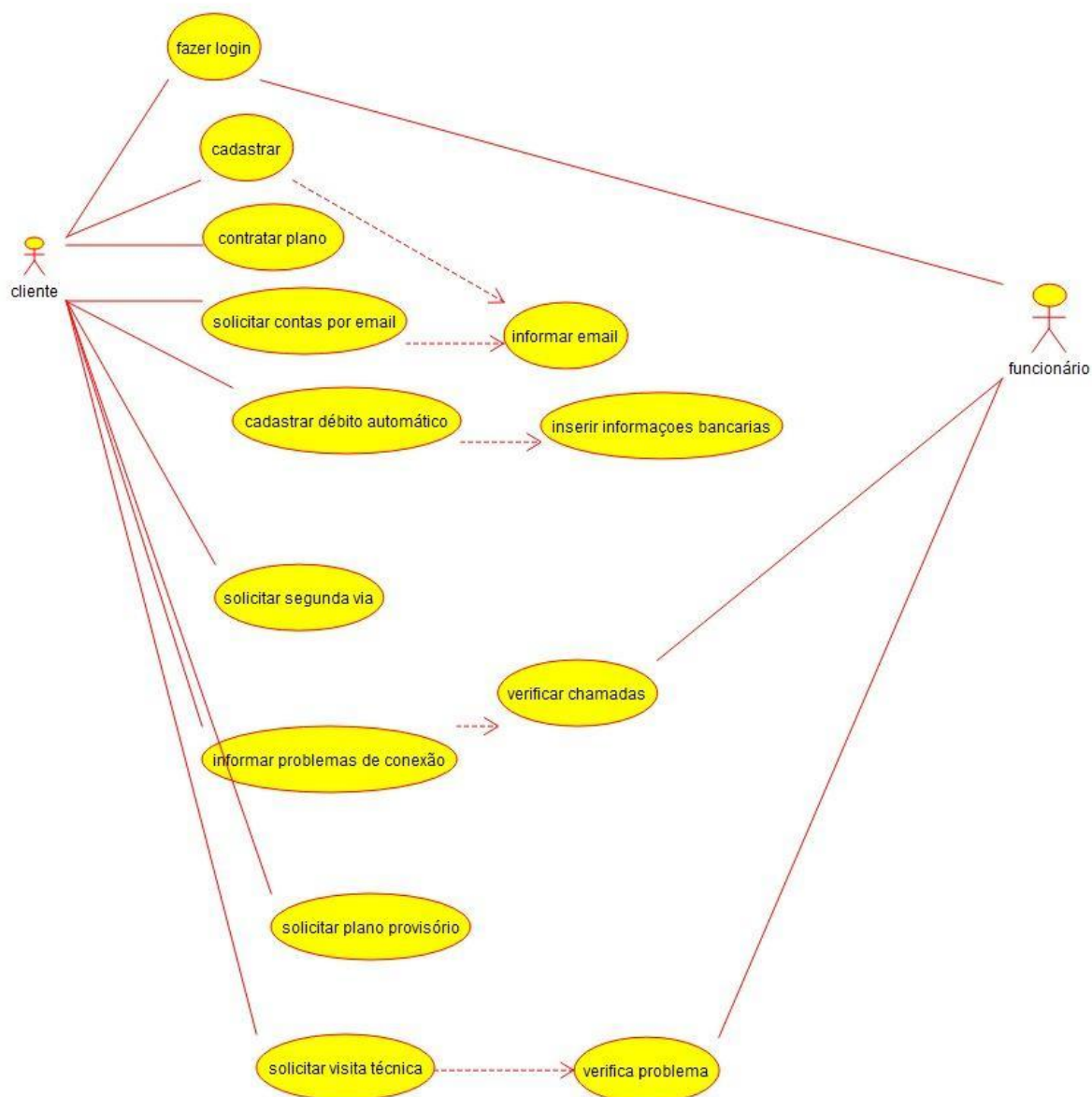


Figura 1: Digrama de casos de uso. Fonte: Autoria própria.

## Tabelas Casos de Uso

ITEM	VALOR
<b>Nome do Caso de Uso</b>	<b>Cadastrar</b>
Atores	Cliente
Finalidade (resumo)	Cadastrar dados no sistema
Descrição (mais detalhes)	O cliente por meio da interface grafica adiciona seus dados ao sistema
Tabelas Manipuladas	

ITEM	VALOR
<b>Nome do Caso de Uso</b>	<b>Cadastrar Imóvel</b>
Atores	Cliente
Finalidade (resumo)	Cadastrar dados no sistema

Descrição (mais detalhes)	O cliente por meio da interface gráfica adiciona dados do seu imóvel ao sistema
Tabelas Manipuladas	

ITEM	VALOR
<b>Nome do Caso de Uso</b>	<b>Solicitar contas por email</b>
Atores	Cliente
Finalidade (resumo)	Cadastrar cadastra o email par receber faturas
Descrição (mais detalhes)	O cliente por meio da interface gráfica adiciona dados referente a sua conta de email para receber as faturas.
Tabelas Manipuladas	

ITEM	VALOR
------	-------

<b>Nome do Caso de Uso</b>	<b>Cadastrar Débito automático</b>
Atores	Cliente
Finalidade (resumo)	Cadastrar dados no sistema
Descrição (mais detalhes)	O cliente por meio da interface gráfica insere a informação de seus meios bancários para pagar as faturas no débito automático.
Tabelas Manipuladas	

<b>ITEM</b>	<b>VALOR</b>
<b>Nome do Caso de Uso</b>	<b>Solicitar Segunda via</b>
Atores	Cliente
Finalidade (resumo)	Solicitar faturas em aberto
Descrição (mais detalhes)	O cliente na interface grafica solicita as faturas em aberto dos seus



	planos.
Tabelas Manipuladas	

ITEM	VALOR
<b>Nome do Caso de Uso</b>	<b>Informar problemas de conexão</b>
Atores	Cliente
Finalidade (resumo)	Informar um problema na conexão
Descrição (mais detalhes)	O cliente consegue informar um problema na sua conexão que ficará registrado no sistema.
Tabelas Manipuladas	

ITEM	VALOR
<b>Nome do Caso de Uso</b>	<b>Solicitar plano provisório</b>

Atores	Cliente
Finalidade (resumo)	Solicitar a liberação da conexão
Descrição (mais detalhes)	O cliente consegue solicitar a liberação de um plano provisório caso esteja com o plano inativo.
Tabelas Manipuladas	

ITEM	VALOR
<b>Nome do Caso de Uso</b>	<b>Solicitar visita técnica</b>
Atores	Cliente,funcionario
Finalidade (resumo)	Solicitar visita para resolver problemas
Descrição (mais detalhes)	O cliente relata algum problema na conexão e o funcionário verifica se é necessário encaminhar um técnico até o local para realizar o reparo.

Tabelas Manipuladas	
---------------------	--

ITEM	VALOR
<b>Nome do Caso de Uso</b>	<b>Fazer Login</b>
Atores	Cliente,funcionario
Finalidade (resumo)	Fazer login no sistema
Descrição (mais detalhes)	O cliente e o funcionário podem fazer login no sistema por meio dos seus usuários cadastrados.
Tabelas Manipuladas	

ITEM	VALOR
<b>Nome do Caso de Uso</b>	<b>Verificar chamados</b>
Atores	Funcionário
Finalidade (resumo)	Verificar chamados abertos pelos usuários



## o Cadastrar

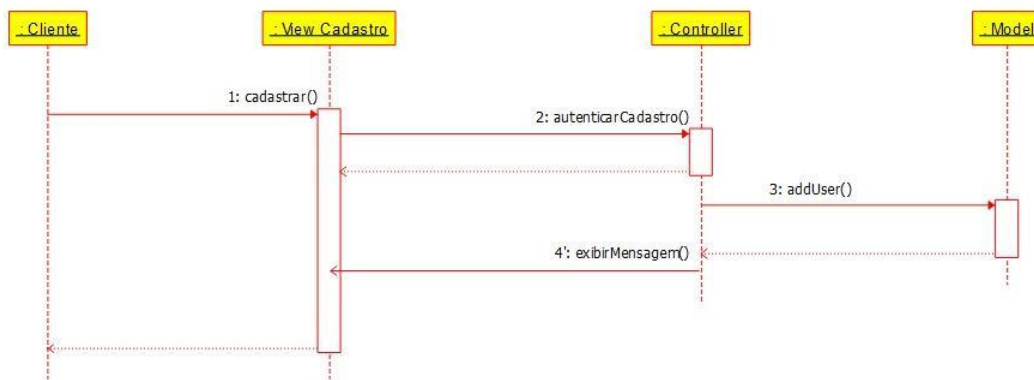


Figura 3:Digrama de Sequência: Cadastrar. Fonte: Autoria própria.

## o Realizar Login

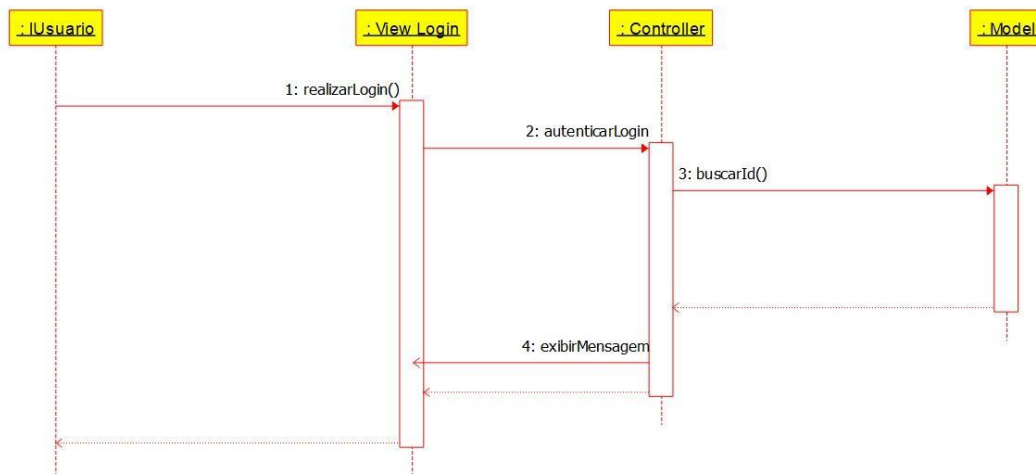


Figura 4:Digrama de Sequência: Realizar Login. Fonte: Autoria própria.

## o Verificar Solicitação

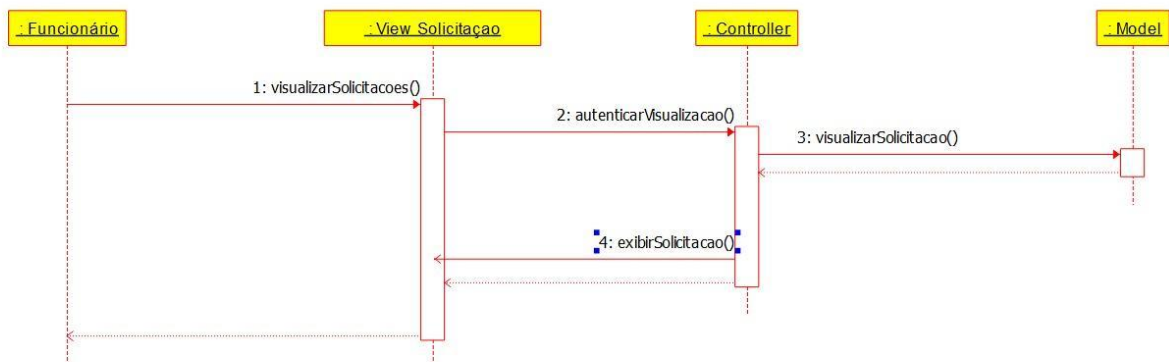


Figura 5:Digrama de Sequência: Verificar Solicitação. Fonte: Autoria própria.

- Diagrama de Estados
  - o Cadastro

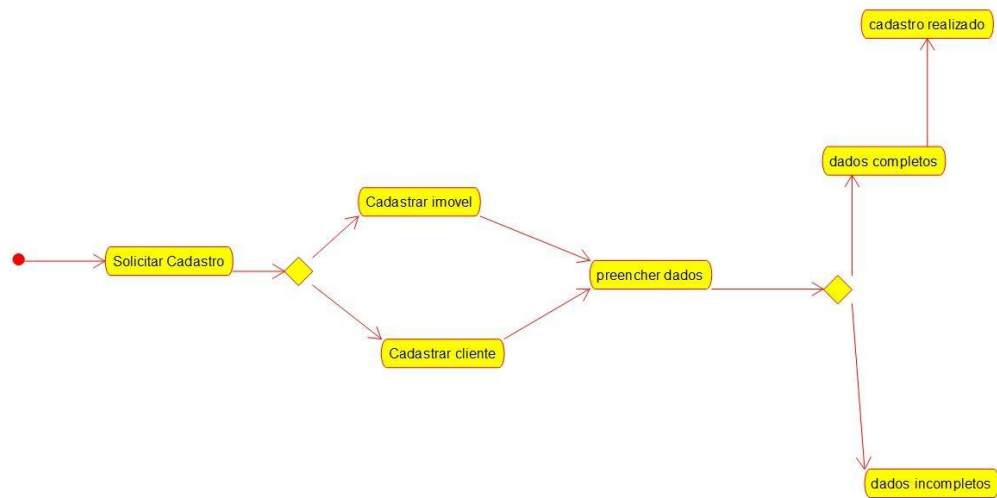


Figura 6:Diagrama de estados: Cadastro. Fonte: Autoria própria.

- o Login

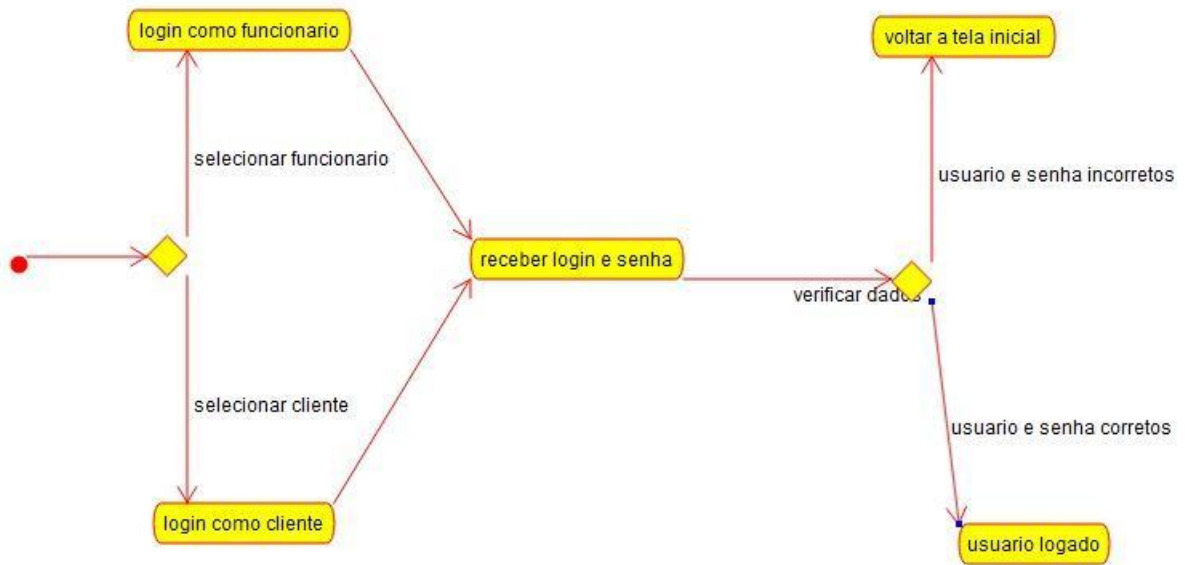


Figura 7:Diagrama de estados: Login. Fonte: Autoria própria.

o Solicitar segunda via

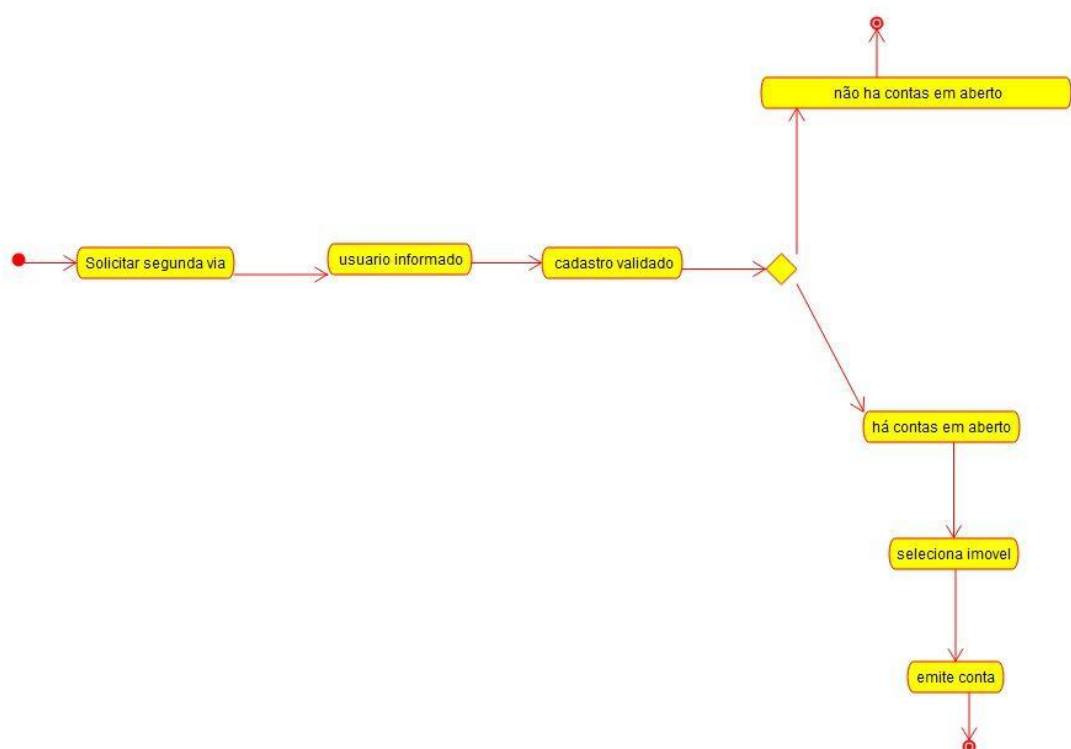


Figura8:Digrama de estados: Solicitar Segunda Via. Fonte: Autoria própria.

#### 4. Conceitos aplicados: Polimorfismo, Padrão de Projeto, Controle de erros e Integração com outro projeto.

No desenvolvimento do projeto, foram implementados os conceitos de polimorfismo, controle de erros e a aplicação do padrão de projeto. O polimorfismo, conforme ilustrado na Figura 9, foi incorporado à JanelaLogin com o propósito de discernir se o acesso seria efetuado como cliente ou funcionário.

```
////////POLIMORFISMO//////////
if (this.controller.validaLogin(login)) {
    int posiUser = this.controller.buscarUser(login);
    Usuario user = (Usuario) this.controller.getUsuario(indice: posiUser);

    if (user instanceof Funcionario) {
        paginas.show(parent: this.painelPrincipal, name: "telaServFuncionario");
    } else if (user instanceof Cliente) {
        Cliente client = (Cliente) user;
        Cliente.setCliente(user: client);
        this.painelPrincipal.add(comp: TelaPlanos.iniciar(layout: paginas), constraints: "telaInfos");
        this.painelPrincipal.add(new TelaCadastroPlano(paginas), constraints: "telaCadastroPlano");
        this.painelPrincipal.add(new TelaInfProblema(layout: paginas), constraints: "telaProblemaConexao");
        paginas.show(parent: this.painelPrincipal, name: "telaServicos");
    }
} else {
    JOptionPane.showMessageDialog(parentComponent: this, message: "Usuário e/ou senha incorretos",
        title: "Login Incorreto", messageType: JOptionPane.ERROR_MESSAGE);
}
////////POLIMORFISMO//////////
```

Figura 9:Polimorfismo JanelaLogin. Fonte: Autoria própria.

O padrão de projeto adotado foi o Singleton, escolha motivada pela sua utilidade na execução do programa ao restringir a implementação das classes Cliente e Controller a apenas uma instância por vez. Tal abordagem minimiza a probabilidade de ocorrência de erros, impedindo que o usuário acesse mais de um cadastro simultaneamente. Além disso, o Singleton foi aplicado nas janelas da interface, garantindo que o usuário não possa abrir a mesma janela em múltiplas instâncias.

```
////////SINGLETON//////////
public static Controller iniciar(){
    if (controller == null)
        controller = new Controller();
    return controller;
}

private Controller(){
    Usuario funcionario = new Funcionario(nome: "Administrador");
    Login loginFuncionario = new Login(idUser: "funcionario", senha: "funcionario123");
    funcionario.setLogin(login: loginFuncionario);
    this.model = new Model();
    this.model.addUsuario(usuario: funcionario);
}
////////SINGLETON//////////
```



Figura 10: Padrão de Projeto Singleton Classe Controller. Fonte: Autoria própria.

```
/////////SINGLETON/////////
public static Cliente iniciar() {
    if (cliente == null)
        cliente = new Cliente();
    return cliente;
}
public static void fechar() {
    cliente = null;
}
public static void setCliente(Cliente user) {
    cliente = user;
}
public static Cliente getCliente() {
    return cliente;
}

/////////SINGLETON/////////
```

Figura 11: Padrão de Projeto Singleton Classe Cliente. Fonte: Autoria própria.

O controle de erros foi predominantemente integrado à classe Controller, visando evitar a inserção de valores divergentes de números nos campos, como CEP ou telefone. Para este propósito, uma exceção verificada foi empregada, conforme exemplificado na Figura 12.

```
//////////Excecao Verificada//////////
public int validaNumeros(String numero) {
    int num;
    try {
        num = Integer.parseInt(s: numero);
    } catch (NumberFormatException e) {
        num = -999;
        System.out.println(x: e.getMessage());
    }
    return num;
}

//////////Excecao verificada//////////
```

Figura 12: Controle de erros. Classe Controller. Fonte: Autoria própria.

Foi esboçada uma ideia de integração com o grupo 8, que implementou um sistema de gerenciamento de um restaurante universitário, onde é feito uma classe para um plano especial de rede para um restaurante universitário. Foram inseridos os dados para que fosse feito o cadastro em nosso sistema.

## **5. Conclusão**

Ao finalizar este projeto em Java para a criação de uma central de autoatendimento para os clientes de uma empresa de internet, ficou evidente o quão importantes são os conceitos aprendidos na disciplina de Programação Orientada a Objetos. Esses conceitos foram essenciais para desenvolver o trabalho que fundamenta este relatório, abrindo caminhos para explorar outras possibilidades no futuro.

## **6. Referências Bibliográficas**

DEITEL, Harvey M.,; DEITEL, Paul J. C++ como programar. 5. ed. Porto Alegre, RS: Pearson Prentice Hall, 2006. xlii, 1163 p. + 1 CD-ROM ISBN 8576050560.

MEYERS, Scott. C++ moderno e eficaz. 1. ed. Rio de Janeiro, RJ: Alta Books, 2016. 366 p. ISBN 9788550800035 (broch.).

DEITEL, Paul J.; DEITEL, Harvey M. Java, como programar. 10. ed. São Paulo, SP: Pearson Prentice Hall, 2017. xxix, 1144 p. ISBN 9788543004792.

H. M. Deitel, P. J. Deitel. Java: Como Programar, 8a. Edição. Pearson, 2010