



**Universidade de Brasília**

**Departamento de Ciência da Computação**

**CIC 0197 - Técnicas de Programação 1**

**Trabalho Prático - Aplicação Governamental para Auxílio a  
Escolas Públicas**

**Professora: Roberta Barbosa Oliveira**

**Autores:**

Lucas Henrique Alves Rosa - 18/0042572

Marcos Eduardo Monteiro Junqueira Junqueira - 18/0023691

Brasília

12 de setembro de 2022

## Sumário

<b>Sumário</b>	<b>2</b>
<b>1 Descrição do Problema</b>	<b>3</b>
1.1 GitHub	3
<b>2 Regras de Negócio</b>	<b>3</b>
2.1 Cadastro	3
2.2 Serviços	3
<b>3 Classes</b>	<b>5</b>
3.1 Diagrama de Classe	5
3.2 GerenciadorDeArquivos	5
3.3 Usuario	6
3.4 Escola	7
3.5 Pedido	8
3.6 DiretorDaEscola	8
3.7 Chamado	11
3.8 Energia	12
3.9 Internet	13
3.10 Computador	14
3.11 EntidadeDoGoverno	16
<b>4 Telas</b>	<b>20</b>
4.1 Tela Inicial	20
4.2 Cadastro Entidade do Governo	20
4.3 Cadastro Diretor	21
4.4 Tela Inicial do Diretor	22
4.5 Tela de pedidos do Diretor	23
4.6 Tela de detalhes de pedidos do Diretor	23
4.7 Trocar senha ou nome de usuário	23
4.8 Tela inicial da entidade	24
4.9 Tela de pedidos	24
4.10 Tela de Chamado de Energia	25
4.11 Tela de Chamado de Internet	25
4.12 Tela de Chamado de Computador	26

## 1 Descrição do Problema

Nos últimos anos, foi possível notar a importância da tecnologia no processo da educação dentro das escolas. Ferramentas como vídeos educativos, jogos educacionais e ferramentas interativas já começaram a surgir como opções que escolas e professores podem trazer para auxiliar o processo educativo e se mostraram consideravelmente efetivas. Além disso, em luz dos eventos recentes, a abertura das escolas de forma remota durante a pandemia de COVID-19 tornou-se possível somente por meio da internet e das múltiplas plataformas disponíveis nela (Ex. Google Classroom, Microsoft Teams, etc.). Dessa forma, é possível perceber a importância crescente que essas tecnologias terão no processo educacional. Nessa luz, esse projeto tem como objetivo fazer um sistema para o cadastramento e o auxílio de escolas públicas nos âmbitos de conectividade e modernização. São oferecidos serviços em formatos de programas governamentais que oferecem internet, energia e computadores através de pedidos feitos pela própria instituição.

### 1.1 GitHub

O link para acessar nosso repositório no gitHub é <https://github.com/LucasHARosa/EscolasConectadas.git>

## 2 Regras de Negócio

### 2.1 Cadastro

No sistema serão cadastrados os usuários que poderão fazer operações no sistema, cada usuário terá um login único e senha. Existem duas opções para realizar o cadastramento, a primeira é como entidade governamental que terá um cargo e sua identificação de funcionário, a segunda opção é como diretor da escola, este por sua vez deve cadastrar suas informações pessoais e a escola na qual ele coordena, o cadastro do diretor no sistema só é possível se os dados da escola forem preenchidos. Entre os dados referentes a escola são importantes os dados referentes ao endereço da escola, e isso inclui sua geolocalização, número de computadores presentes, quantidade de alunos, se a escola possui conexão com a internet ou não e se possui energia elétrica.

### 2.2 Serviços

Uma vez feito o cadastro do diretor e da escola o diretor pode fazer o login no sistema com seus dados cadastrados e dentro do sistema ele pode fazer até 3 pedidos diferentes, independente da situação da escola. O pedido pode ser para solicitar conexão com internet, instalação de energia elétrica ou até computadores para os alunos, no pedido não são observadas as condições da escola, ou seja, se a escola que está solicitando internet tem ou não conexão. Além disso, os diretores podem observar os pedidos que estão em processamento e os pedidos que foram aprovados em uma página separada, bem como as observações associadas a ele. Pedidos recusados pela entidade são automaticamente excluídos e é possível realizar somente um pedido de cada tipo, sendo necessário excluir o pedido do tipo anterior caso deseje realizar um novo do mesmo tipo.

Já a entidade governamental pode olhar a lista de pedidos feitos pelos diretores das escolas e pode tratar cada pedido individualmente e mudar seu estado interno de não apto para apto. A entidade deve verificar se o pedido está apto para ser aceito ou não, só assim poderá criar um chamado para a realização de um dos 3 serviços. É possível mudar o status da escola através da entidade do governo, assim que um pedido é aceito e o chamado é realizado, o status da escola deve ser mudado para se adequar a nova atualização realizada pelo programa governamental. Os três tipos de chamados são:

- **Serviço de computadores:** Tem como objetivo levar computadores para as escolas e existe uma relação entre o número de computadores em relação a quantidade de matrículas, e uma condição extra é que a escola deve ter internet para o serviço ser feito, caso não tenha a entidade do governo deve solicitar um novo chamado, porém para o serviço de instalação de internet na escola.
- **Serviço de internet:** Tem como objetivo levar internet para a escola e tem como pré-requisito a escola não ter internet e possuir energia, caso não possua, deve ser realizado um novo chamado.
- **Serviço de energia:** Tem como objetivo levar energia a escola e tem como pré-requisito não ter energia na escola.

Por fim, como funcionalidade adicional é possível mudar o nome do usuário mostrado ao longo da aplicação e a senha, tanto para o diretor como para a entidade.

### 3 Classes

#### 3.1 Diagrama de Classe

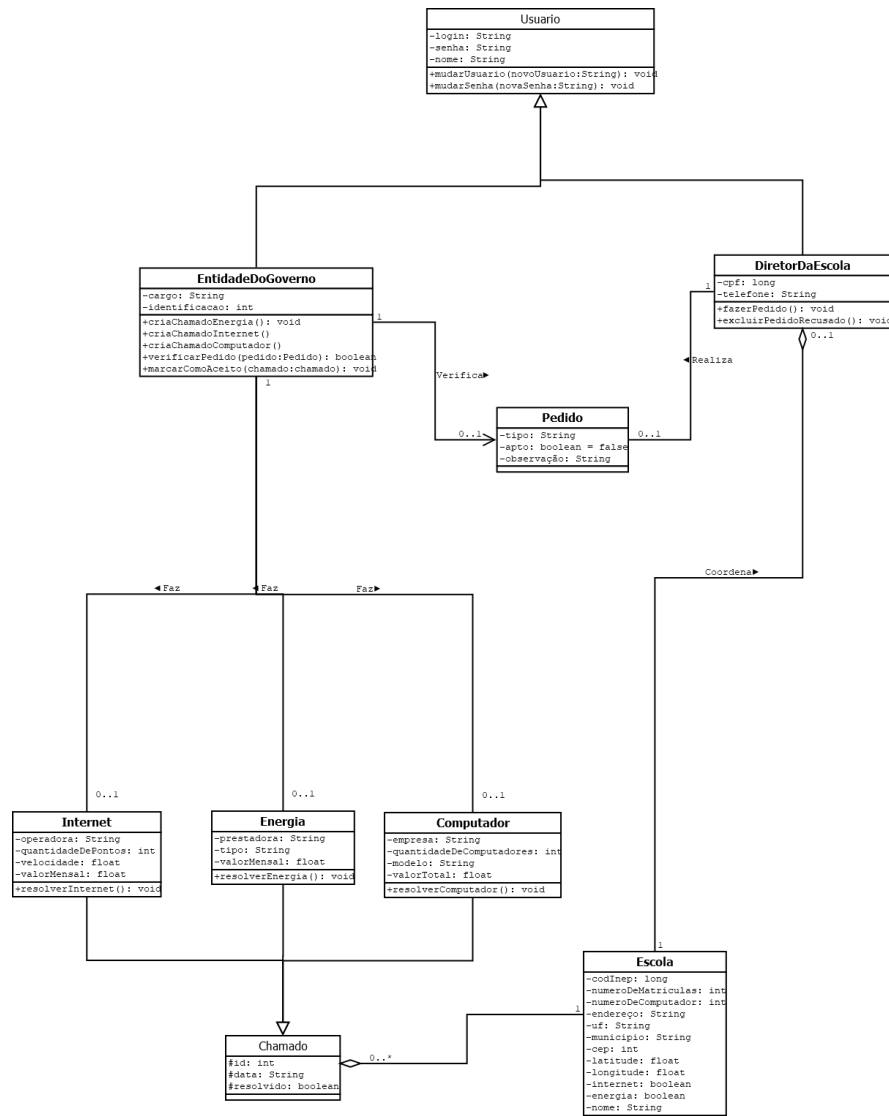


Figura 1 – UML

Na figura acima se apresenta o diagrama de classes do projeto, contendo todas as classes utilizadas, bem como seus atributos e métodos. Cada uma dessas classes e suas implementações serão discutidas em maior aprofundamento nas seções a seguir.

#### 3.2 GerenciadorDeArquivos

```
public class GerenciadorDeArquivos<T>{
    public ArrayList<T> lerArquivo(File arquivo) throws IOException, ClassNotFoundException{
        T obj;
        ArrayList<T> listaDeItens = new ArrayList<>();
```

```

    try (FileInputStream fi = new FileInputStream(arquivo); ObjectInputStream oi = new
        ObjectInputStream(fi)) {
        while (fi.available() > 0) {
            obj = (T) oi.readObject();
            listaDeItens.add(obj);
        }
        oi.close();
        fi.close();
        EscreverArquivo(listaDeItens, arquivo);
        return listaDeItens;
    } catch (IOException | ClassNotFoundException e) {
        throw e;
    }
}

public void EscreverArquivo(ArrayList<T> listaDeItens, File arquivo) throws IOException{
    try (FileOutputStream fs = new FileOutputStream(arquivo); ObjectOutputStream os = new
        ObjectOutputStream(fs)) {
        for (T obj : listaDeItens) {
            os.writeObject(obj);
        }
    } catch (IOException e) {
        throw e;
    }
}
}

```

---

Essa classe é a classe responsável por realizar todas as operações de sobre os arquivos de texto nos quais as informações da aplicação são salvas. Durante a sua inicialização é necessário repassar qual o tipo de objeto será trabalhado, similarmente a declaração de uma ArrayList, e ela apresenta dois métodos distintos: **lerArquivo** e **EscreverArquivo**. O método lerArquivo recebe um arquivo do tipo File indicando qual o documento deve ser lido, ele percorre o arquivo inteiro e retorna uma ArrayList do tipo que foi inicializado com todos os objetos salvos no arquivo. Já EscreverArquivo recebe um arquivo do tipo File indicando qual o documento deve ser escrito e uma ArrayList do tipo inicializado, em seguida essa lista é escrita em sua completude dentro do arquivo de texto fornecido, utilizando ObjectOutputStream.

### 3.3 Usuario

---

```

public abstract class Usuario implements Serializable{
    private static final long serialVersionUID = 1L;
    private String login;
    private String senha;
    private String nome;

    public Usuario(String login, String senha, String nome) {
        this.login = login;
        this.senha = senha;
    }
}

```

```

        this.nome = nome;
    }

    //Getters e Setters da classe

    abstract public void mudarNome(String nome) throws IOException, ClassNotFoundException;

    abstract public void mudarSenha(String senha) throws IOException, ClassNotFoundException;
}

```

---

Essa classe abstrata é responsável por definir as características comuns entre os dois tipos de usuários: **EntidadeDoGoverno** e **DiretorDaEscola**. Essa classe também implementa a interface `Serializable` e o atributo `serialVersionUID` a fim de possibilitar sua leitura e escrita em arquivos de text utilizando `ObjectStreams`. Além dos getters e setters padrões, essa classe implementa dois metodos abstratos para a mudança de senha e de nome de usuário, que vão se diferenciar de acordo com o tipo de usuário. Vale ressaltar que o login do usuário nunca é alterado e somente lido e, por isso, apresenta somente um getter.

### 3.4 Escola

---

```

public class Escola implements Serializable{
    private static final long serialVersionUID = 1L;
    private long codInep;
    private int numMatriculas;
    private int numComputador;
    private String endereco;
    private String uf;
    private String municipio;
    private long cep;
    private float latitude;
    private float longitude;
    private boolean internet;
    private boolean energia;
    private DiretorDaEscola diretor = null;
    private String nome;

    public Escola(long codInep, int numMatriculas, int numComputador, String endereco, String uf,
        String municipio, long cep, float latitude, float longitude, boolean internet, boolean
        energia, String nome) {
        this.codInep = codInep;
        this.numMatriculas = numMatriculas;
        this.numComputador = numComputador;
        this.endereco = endereco;
        this.uf = uf;
        this.municipio = municipio;
        this.cep = cep;
        this.latitude = latitude;
        this.longitude = longitude;
    }
}

```

```

        this.internet = internet;
        this.energia = energia;
        this.nome = nome;
    }

    //Getters e Setters da classe
}

```

---

Essa classe tem somente o objetivo de modelar as características do modelo de escola implementado pela aplicação. Dessa forma, essa classe apresenta somente as informações necessárias acerca da escola, além do seus getters e setters padrões para todos os seu atributos, os quais foram omitidos acima em função do seu grande número e do espaço que ocupam.

### 3.5 Pedido

---

```

public class Pedido implements Serializable{
    private static final long serialVersionUID = 1L;
    private String tipo;
    private boolean apto;
    private String obs;
    private DiretorDaEscola diretorDaEscola;

    public Pedido(String tipo, DiretorDaEscola diretorDaEscola, String obs) {
        this.tipo = tipo;
        this.obs = obs;
        this.diretorDaEscola = diretorDaEscola;
    }

    //Getters e Setters da classe

    public void atualizacao (boolean apto,String obs1){
        this.apto = apto;
        String barra = "| ";
        String obs2 = this.obs.concat(barra).concat(obs1);
        setObs(obs2);
    }
}

```

---

Essa classe tem o objetivo de modelar as características do modelo de pedido implementado pela aplicação. Além dos getters e setters padrões de todos os seu atributos, ele apresenta também um método **atualizacao**, responsável por atualizar o estado de um pedido após ser processado por uma entidade governamental na aplicação. Ele recebe o novo status de apto e uma string com uma observação. Em seguida, o status de apto é alterado para o status repassado e a observação passada é concatenada na observação original do pedido.

### 3.6 DiretorDaEscola

---



```

public class DiretorDaEscola extends Usuario implements Serializable{
private static final long serialVersionUID = 1L;
private long cpf;
private String telefone;
private Escola escola;
private ArrayList<Pedido> pedidos = new ArrayList<>();

public DiretorDaEscola(long cpf, String telefone, String login, String senha, String nome,
    Escola escola) {
    super(login, senha, nome);
    this.cpf = cpf;
    this.telefone = telefone;
    this.escola = escola;
}

```

//Getters e Setters da classe

@Override

```

public void mudarNome(String nome) throws IOException, ClassNotFoundException{
    try {
        File arquivo = new File("src\\dados\\usuarios\\diretores.txt");
        setNome(nome);
        GerenciadorDeArquivos<DiretorDaEscola> gerenciadorDeArquivos = new
            GerenciadorDeArquivos<>();
        ArrayList<DiretorDaEscola> diretores = gerenciadorDeArquivos.lerArquivo(arquivo);
        for (int i = 0; i < diretores.size(); i++) {
            DiretorDaEscola diretor = diretores.get(i);
            if (this.getLogin().equals(diretor.getLogin()) &&
                this.cpf == diretor.getCpf()) {
                diretores.set(i, this);
                gerenciadorDeArquivos.EscreverArquivo(diretores, arquivo);
            }
        }
    } catch (IOException | ClassNotFoundException e) {
        throw e;
    }
}

```

@Override

```

public void mudarSenha(String senha) throws IOException, ClassNotFoundException {
    try {
        File arquivo = new File("src\\dados\\usuarios\\diretores.txt");
        setSenha(senha);
        GerenciadorDeArquivos<DiretorDaEscola> gerenciadorDeArquivos = new
            GerenciadorDeArquivos<>();
        ArrayList<DiretorDaEscola> diretores = gerenciadorDeArquivos.lerArquivo(arquivo);
        for (int i = 0; i < diretores.size(); i++) {
            DiretorDaEscola diretor = diretores.get(i);

```

```

        if (this.getLogin().equals(diretor.getLogin()) &&
            this.cpf == diretor.getCpf()) {
            diretores.set(i, this);
            gerenciadorDeArquivos.EscreverArquivo(diretores, arquivo);
        }
    }
} catch (IOException | ClassNotFoundException e) {
    throw e;
}

}

public ArrayList<Pedido> getPedidos() {
    return this.pedidos;
}

public boolean fazerPedido(Pedido novoPedido) throws IOException, ClassNotFoundException{
    try {
        for (Pedido pedido : pedidos) {
            if(pedido.getTipo().equals(novoPedido.getTipo())){
                return false;
            }
        }
        File arquivo = new File("src\\dados\\usuarios\\diretores.txt");
        pedidos.add(novoPedido);
        GerenciadorDeArquivos<DiretorDaEscola> gerenciadorDeArquivos = new
            GerenciadorDeArquivos<>();
        ArrayList<DiretorDaEscola> diretores = gerenciadorDeArquivos.lerArquivo(arquivo);
        for (int i = 0; i < diretores.size(); i++) {
            DiretorDaEscola diretor = diretores.get(i);
            if (this.getLogin().equals(diretor.getLogin()) &&
                this.cpf == diretor.getCpf()) {
                diretores.set(i, this);
                gerenciadorDeArquivos.EscreverArquivo(diretores, arquivo);
            }
        }
        return true;
    } catch (IOException | ClassNotFoundException e) {
        throw e;
    }
}

public void removerPedido(int indice) throws IOException, ClassNotFoundException{
    try {
        File arquivo = new File("src\\dados\\usuarios\\diretores.txt");
        pedidos.remove(indice);
        GerenciadorDeArquivos<DiretorDaEscola> gerenciadorDeArquivos = new
            GerenciadorDeArquivos<>();
        ArrayList<DiretorDaEscola> diretores = gerenciadorDeArquivos.lerArquivo(arquivo);
    }
}

```

```

        for (int i = 0; i < diretores.size(); i++) {
            DiretorDaEscola diretor = diretores.get(i);
            if (this.getLogin().equals(diretor.getLogin()) &&
                this.cpf == diretor.getCpf()) {
                diretores.set(i, this);
                gerenciadorDeArquivos.EscreverArquivo(diretores, arquivo);
            }
        }
    } catch (IOException | ClassNotFoundException e) {
        throw e;
    }
}

```

---

Essa classe tem como objetivo modelar o comportamento de um usuário do tipo diretor. De forma análoga a citada acima, essa implementa a interface `Serializable` e o atributo `serialVersionUID` a fim de possibilitar sua leitura e escrita em arquivos de texto utilizando `ObjectStreams`, além de herdar da classe de Usuário descrita acima. Além dos getter e setter padrões, essa classe apresenta os seguintes métodos:

- **mudarNome:** Override do método abstrato implementado pela classe de usuário. Ela recebe um novo nome de usuário, o qual é modificado no objeto pelo método herdado de `setNome`. Em seguida, é instanciado um `GerenciadorDeArquivos` que retorna a `ArrayList` com todos os diretores cadastrados. Essa `ArrayList` é então atualizada com o novo nome na posição correspondente ao diretor em questão e essa `ArrayList` atualizada é salva no arquivo por meio do método `EscreverArquivo` do `GerenciadorDeArquivos`.
- **mudarSenha:** Override do método abstrato implementado pela classe de usuário. Ela recebe uma nova senha, a qual é modificada no objeto pelo método herdado de `setSenha`. Em seguida, é instanciado um `GerenciadorDeArquivos` que retorna a `ArrayList` com todos os diretores cadastrados. Essa `ArrayList` é então atualizada com a nova senha na posição correspondente ao diretor em questão e essa `ArrayList` atualizada é salva no arquivo por meio do método `EscreverArquivo` do `GerenciadorDeArquivos`.
- **fazerPedido:** Método utilizado para adicionar pedido na lista de Pedidos realizados pelo diretor. Após receber o novo pedido como parâmetro, é realizada uma verificação se um pedido de mesmo tipo já foi realizado, em seguida é instanciado um `GerenciadorDeArquivos` que retorna a `ArrayList` com todos os diretores cadastrados. Essa `ArrayList` é então atualizada com o novo pedido na posição correspondente ao diretor em questão e essa `ArrayList` atualizada é salva no arquivo por meio do método `EscreverArquivo` do `GerenciadorDeArquivos`. O método retorna falso em caso do novo pedido ser de um tipo que já existe na lista de pedidos do diretor e retorna verdadeiro em caso do pedido ter sido salvo com sucesso no arquivo.
- **removerPedido:** Método responsável por remover um determinado pedido da lista de pedidos realizados pelo diretor. Esse método recebe o índice do pedido em questão, que é então removido da `ArrayList` de pedidos. Em seguida, é realizado o processo de atualização do arquivo de diretores análogo ao descrito acima pelos demais métodos.

### 3.7 Chamado

---

```

public abstract class Chamado implements Serializable {
    private static final long serialVersionUID = 1L;
    protected int id;
    protected String data;
    protected boolean resolvido;
    protected Escola escola;

    public Chamado(int id, String data, Escola escola) {
        this.id = id;
        this.data = data;
        this.escola = escola;
    }

    //Getters e Setters da classe

    public abstract int resolverChamado();
}

```

---

Essa classe abstrata é responsável para definir as características comuns entre os chamados de **Energia**, **Internet** e **Computadores**. Essa classe também implementa a interface `Serializable` e o atributo `serialVersionUID` a fim de possibilitar sua leitura e escrita em arquivos de text utilizando `ObjectStreams`. Além dos getters e setters padrões, essa classe implementa um método abstrato para resolver o chamado que analisa os atributos relacionados aos chamados e a escola.

### 3.8 Energia

---

```

public class Energia extends Chamado implements Serializable {
    private static final long serialVersionUID = 1L;
    private String prestadora;
    private String tipo;
    private float valorMensal;

    public Energia(String prestadora, String tipo, float valorMensal, int id, String data, Escola
        escola) {
        super(id, data, escola);
        this.prestadora = prestadora;
        this.tipo = tipo;
        this.valorMensal = valorMensal;
    }

    //Getters e Setters da classe

    @Override

```

```

public int resolverChamado(){
    if(this.getEscola().isEnergia()==false){
        setResolvido(true);
        JOptionPane.showMessageDialog(null,"O chamado foi resolvido e est em
            ordem", "Mensagem",JOptionPane.PLAIN_MESSAGE);
        this.getEscola().setEnergia(true);
    }
    else{
        JOptionPane.showMessageDialog(null,"O chamado no poder ser concluso a escola j tem
            energia", "Mensagem",JOptionPane.PLAIN_MESSAGE);
    }

    return 0;
}
}

```

---

Essa classe herda atributos e métodos da classe pai **Chamado** e tem como objetivo modelar as características da energia na escola. Essa classe também implementa a interface `Serializable` e o atributo `serialVersionUID` a fim de possibilitar sua leitura e escrita em arquivos de texto utilizando `ObjectStreams`. Além dos getters e setters padrões, essa classe implementa um método abstrato para resolver o chamado de energia.

- **resolveChamado()**: Método que consiste em verificar a situação da escola em relação a energia, modificar seu atributo *Resolvido* e exibir uma mensagem ao usuário em relação a situação.

### 3.9 Internet

---

```

public class Internet extends Chamado{
    private String operadora;
    private int quantidadePontos;
    private float velocidade;
    private float valoMensal;

    public Internet(String operadora, int quantidadePontos, float velocidade, float valoMensal, int
        id, String data, Escola escola) {
        super(id, data, escola);
        this.operadora = operadora;
        this.quantidadePontos = quantidadePontos;
        this.velocidade = velocidade;
        this.valoMensal = valoMensal;
    }

    //Getters e Setters da classe

    @Override
    public int resolverChamado(){

```

```

int opcao=0;
if(this.getEscola().isInternet()==false){
    setResolvido(true);
    if(this.getEscola().isEnergia()==false){
        opcao=10;
    }
    JOptionPane.showMessageDialog(null,"O chamado foi resolvido e est em
        ordem", "Mensagem", JOptionPane.PLAIN_MESSAGE);
    this.getEscola().setInternet(true);
}
else{
    JOptionPane.showMessageDialog(null,"O chamado no poder ser concluido a escola j tem
        Internet", "Mensagem", JOptionPane.PLAIN_MESSAGE);
}
return opcao;
}
}

```

---

Essa classe herda atributos e métodos da classe pai **Chamado** e tem como objetivo modelar as características da internet na escola. Essa classe também implementa a interface `Serializable` e o atributo `serialVersionUID` a fim de possibilitar sua leitura e escrita em arquivos de texto utilizando `ObjectStreams`. Além dos getters e setters padrões, essa classe implementa um método abstrato para resolver o chamado de internet.

- **resolveChamado()** Método que consiste em verificar a situação da escola em relação a internet e energia. Caso a escola tenha energia seu fluxo de trabalho consiste em verificar a situação da escola em relação a internet, modificar seu atributo *Resolvido* e exibir uma mensagem ao usuário em relação a situação. Caso a escola não possua internet e nem energia a variável de retorno sinaliza isso e é criado mais um chamado para a escola, que é de energia. Esse chamado é criado da seguinte forma:

```

if(opcao == 10){
    JOptionPane.showMessageDialog(null,"O chamado de Energia foi criado"
        , "Mensagem", JOptionPane.PLAIN_MESSAGE);
    entidade.criaChamadoEnergia(internet.getEscola());
}

```

---

### 3.10 Computador

---

```

public class Computador extends Chamado {
    private String empresa;
    private int quantComputador;
    private String modelo;
    private float valorTotal;
}

```

```

public Computador(String empresa, int quantComputador, String modelo, float valorTotal, int id,
    String data, Escola escola) {
    super(id, data, escola);
    this.empresa = empresa;
    this.quantComputador = quantComputador;
    this.modelo = modelo;
    this.valorTotal = valorTotal;
}

//Getters e Setters da classe

@Override
public int resolverChamado(){
    int opcao=0;
    if(this.quantComputador>0){
        setResolvido(true);
        if(this.getEscola().isEnergia()==false){
            opcao=10;
        }
        if(this.getEscola().isInternet()==false){
            opcao++;
        }
        JOptionPane.showMessageDialog(null,"0 chamado foi resolvido e est em
            ordem", "Mensagem", JOptionPane.PLAIN_MESSAGE);
        this.getEscola().setNumComputador(this.getEscola().getNumMatriculas());
    }
    else{
        JOptionPane.showMessageDialog(null,"0 chamado no poder ser concluido a escola j tem
            Computadores", "Mensagem", JOptionPane.PLAIN_MESSAGE);
    }
    return opcao;
}
}

```

---

Essa classe herda atributos e métodos da classe pai **Chamado** e tem como objetivo modelar as características de computadores na escola. Essa classe também implementa a interface `Serializable` e o atributo `serialVersionUID` a fim de possibilitar sua leitura e escrita em arquivos de texto utilizando `ObjectStreams`. Além dos getters e setters padrões, essa classe implementa um método abstrato para resolver o chamado de computadores.

- **resolveChamado()**: Método que consiste em verificar a situação da escola em relação a quantidade de computadores, internet e energia. Caso a escola tenha energia e internet seu fluxo de trabalho consiste em verificar a situação da escola em relação a quantidade de computadores, modificar seu atributo *Resolvido* e exibir uma mensagem ao usuário em relação a situação. Caso a escola não possua internet e nem energia a variável de retorno sinaliza isso e é criado mais dois chamado para a escola, que é de energia e internet. Esses chamados são criados da seguinte forma:
-

```

if(computador.isResolvido()==false){
    opcao = computador.resolverChamado();
    if(opcao >=10){
        JOptionPane.showMessageDialog(null,"O chamado de Energia foi
            criado","Mensagem",JOptionPane.PLAIN_MESSAGE);
        entidade.criaChamadoEnergia(computador.getEscola());
    }
    else if(opcao == 11 || opcao == 1){
        JOptionPane.showMessageDialog(null,"O chamado de Internet foi
            criado","Mensagem",JOptionPane.PLAIN_MESSAGE);

        entidade.criaChamadoComputador(computador.getEscola());
    }
}

```

---

### 3.11 EntidadeDoGoverno

---

```

public class EntidadeDoGoverno extends Usuario implements Serializable{
    private static final long serialVersionUID = 1L;
    private String cargo;
    private int identificacao;

    public EntidadeDoGoverno(String cargo, int identificacao, String login, String senha, String
        nome) {
        super(login, senha, nome);
        this.cargo = cargo;
        this.identificacao = identificacao;
    }

    //Getters e Setters da classe

    @Override
    public String toString(){
        return "Login:" + this.getLogin() + "\nSenha: " + this.getSenha();
    }

    @Override
    public void mudarNome(String nome) throws IOException, ClassNotFoundException {
        try {
            File arquivo = new File("src\\dados\\usuarios\\entidades.txt");
            setNome(nome);
            GerenciadorDeArquivos<EntidadeDoGoverno> gerenciadorDeArquivos = new
                GerenciadorDeArquivos<>();
            ArrayList<EntidadeDoGoverno> entidades = gerenciadorDeArquivos.lerArquivo(arquivo);
            for (int i = 0; i < entidades.size(); i++) {
                EntidadeDoGoverno entidade = entidades.get(i);
            }
        }
    }
}

```



```

        if (this.getLogin().equals(entidade.getLogin()) &&
            this.identificacao == entidade.getIdentificacao()) {
            entidades.set(i, this);
            gerenciadorDeArquivos.EscriverArquivo(entidades, arquivo);
        }
    }
} catch (IOException | ClassNotFoundException e) {
    throw e;
}
}

@Override
public void mudarSenha(String senha) throws IOException, ClassNotFoundException {
    try {
        File arquivo = new File("src\\dados\\usuarios\\entidades.txt");
        setSenha(senha);
        GerenciadorDeArquivos<EntidadeDoGoverno> gerenciadorDeArquivos = new
            GerenciadorDeArquivos<>();
        ArrayList<EntidadeDoGoverno> entidades = gerenciadorDeArquivos.lerArquivo(arquivo);
        for (int i = 0; i < entidades.size(); i++) {
            EntidadeDoGoverno entidade = entidades.get(i);
            if (this.getLogin().equals(entidade.getLogin()) &&
                this.identificacao == entidade.getIdentificacao()) {
                entidades.set(i, this);
                gerenciadorDeArquivos.EscriverArquivo(entidades, arquivo);
            }
        }
    } catch (IOException | ClassNotFoundException e) {
        throw e;
    }
}

public void criaChamadoEnergia(Escola escola) throws IOException, ClassNotFoundException {
    Energia novoEnergia = new Energia("Provedora De Energia Local",
        "Rede pblica",
        1200,
        1,
        "Outubro",
        escola);

    novoEnergia.setResolvido(false);
    try {
        File arquivo = new File("src\\dados\\usuarios\\chamadosEnergia.txt");

        GerenciadorDeArquivos<Energia> gerenciadorDeArquivos = new GerenciadorDeArquivos<>();
        ArrayList<Energia> energia = new ArrayList<>();
        if (!arquivo.isDirectory() && arquivo.exists()) {
            energia = gerenciadorDeArquivos.lerArquivo(arquivo);
        }
    }
}

```

```

        energia.add(novoEnergia);
        gerenciadorDeArquivos.EscreverArquivo(energia, arquivo);
    } catch (IOException | ClassNotFoundException e) {
        JOptionPane.showMessageDialog(null, "Ocorreu um Erro ao abrir o arquivo aqui (" +
            e.toString() + ") " , "ERRO" , JOptionPane.ERROR_MESSAGE);
    }
}

public void criaChamadoInternet(Escola escola) throws IOException, ClassNotFoundException{
    int quantidadePontos = escola.getNumMatriculas()/10;
    Internet novoInternet = new Internet("Provedora De Internet Local",
        quantidadePontos,
        100,
        quantidadePontos*150,
        2,
        "Outubro",
        escola);

    novoInternet.setResolvido(false);
    try {
        File arquivo = new File("src\\dados\\usuarios\\chamadosInternet.txt");

        GerenciadorDeArquivos<Internet> gerenciadorDeArquivos = new GerenciadorDeArquivos<>();
        ArrayList<Internet> internet = new ArrayList<>();
        if (!arquivo.isDirectory() && arquivo.exists()) {
            internet = gerenciadorDeArquivos.lerArquivo(arquivo);
        }
        internet.add(novoInternet);
        gerenciadorDeArquivos.EscreverArquivo(internet, arquivo);
    } catch (IOException | ClassNotFoundException e) {
        JOptionPane.showMessageDialog(null, "Ocorreu um Erro ao abrir o arquivo aqui (" +
            e.toString() + ") " , "ERRO" , JOptionPane.ERROR_MESSAGE);
    }
}

public void criaChamadoComputador(Escola escola) throws IOException, ClassNotFoundException{
    int quantidadeComputador = escola.getNumMatriculas()-escola.getNumComputador();
    Computador novoComputador;
    if(quantidadeComputador<=0){
        quantidadeComputador = 0;
        novoComputador = new Computador("Nenhum",
            quantidadeComputador,
            "Nenhum",
            0,
            3,
            "Outubro",
            escola);
    }
}

```

```

else{
    novoComputador = new Computador("Empresa local",
        quantidadeComputador,
        "Marca",
        quantidadeComputador*1000,
        3,
        "Outubro",
        escola);
}
novoComputador.setResolvido(false);
try {
    File arquivo = new File("src\\dados\\usuarios\\chamadosComputador.txt");

    GerenciadorDeArquivos<Computador> gerenciadorDeArquivos = new GerenciadorDeArquivos<>();
    ArrayList<Computador> computador = new ArrayList<>();
    if (!arquivo.isDirectory() && arquivo.exists()) {
        computador = gerenciadorDeArquivos.lerArquivo(arquivo);
    }
    computador.add(novoComputador);
    gerenciadorDeArquivos.EscreverArquivo(computador, arquivo);
} catch (IOException | ClassNotFoundException e) {
    JOptionPane.showMessageDialog(null,"Ocorreu um Erro ao abrir o arquivo aqui (" +
        e.toString() + ")","ERRO",JOptionPane.ERROR_MESSAGE);
}
}
}

```

Essa classe tem como objetivo modelar o comportamento de um usuário do tipo entidade do governo. Essa classe implementa a interface Serializable e o atributo serialVersionUID a fim de possibilitar sua leitura e escrita em arquivos de texto utilizando ObjectStreams, além de herdar da classe de Usuário descrita acima. Além dos getter e setter padrões, essa classe apresenta os seguintes métodos:

- **mudarNome:** Override do método abstrato implementado pela classe de usuário. Ela recebe um novo nome de usuário, o qual é modificado no objeto pelo método herdado de setNome. Em seguida, é instanciado um GerenciadorDeArquivos que retorna a ArrayList como todos os diretores cadastrados. Essa ArrayList é então atualizada com o novo nome na posição correspondente ao diretor em questão e essa ArrayList atualizada é salva no arquivo por meio do método EscreverArquivo do GerenciadorDeArquivos.
- **mudarSenha:** Override do método abstrato implementado pela classe de usuário. Ela recebe uma nova senha, a qual é modificada no objeto pelo método herdado de setSenha. Em seguida, é instanciado um GerenciadorDeArquivos que retorna a ArrayList como todos os diretores cadastrados. Essa ArrayList é então atualizada com a nova senha na posição correspondente ao diretor em questão e essa ArrayList atualizada é salva no arquivo por meio do método EscreverArquivo do GerenciadorDeArquivos.
- **criaChamadoEnergia:** Método utilizado para adicionar um chamado na lista de chamados de energia e escrever no arquivo que contém essa lista. É criado um novo objeto do tipo energia com seus

parâmetros. Esse método recebe a escola que vai ser realizado o pedido para posteriormente realizar a resolução do pedido.

- **criaChamadoInternet:** Método utilizado para adicionar um chamado na lista de chamados de internet e escrever no arquivo que contém essa lista. É criado um novo objeto do tipo internet com seus parâmetros. Esse método recebe a escola que vai ser realizado o pedido para posteriormente realizar a resolução do pedido.
- **criaChamadoComputador:** Método utilizado para adicionar um chamado na lista de chamados de computador e escrever no arquivo que contém essa lista. É criado um novo objeto do tipo computador com seus parâmetros, o chamado tem duas formas de ser criado de acordo com a quantidade de computadores necessários. Esse método recebe a escola que vai ser realizado o pedido para posteriormente realizar a resolução do pedido.

## 4 Telas

### 4.1 Tela Inicial

Primeira tela do programa, nela é possível acessar as telas de cadastro dos dois tipos de usuários e entrar no programa usando login e senhas cadastradas. Caso a alguma informação do login esteja errada o usuário será avisado.



Figura 2 – Tela Inicial

### 4.2 Cadastro Entidade do Governo

Tela do programa que realiza o cadastro da entidade governamental. Caso a alguma informação do login esteja errada ou exista um o usuário com mesmo login o usuário será avisado.



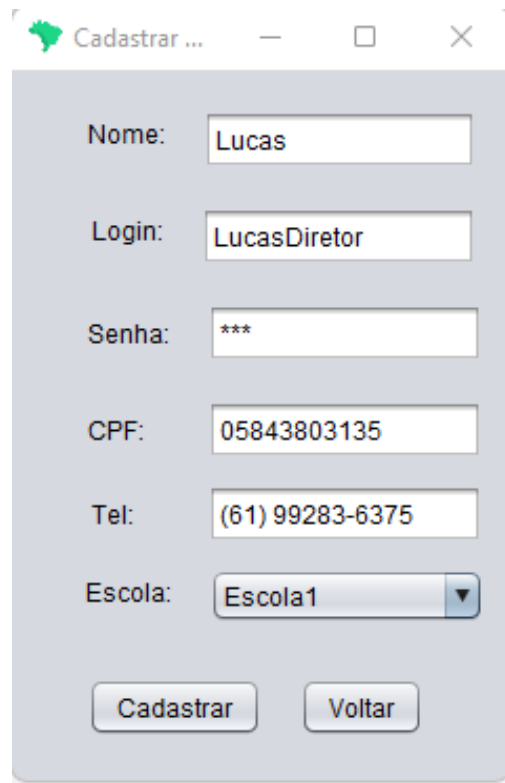
A screenshot of a software window titled "Cadastrar ...". The window has a light gray background and contains five text input fields arranged vertically. Each field is preceded by a label: "Nome:", "Login:", "Senha:", "Cargo:", and "Identificação:". The inputs contain the following values: "Marcos", "MarcosChefe", "\*\*\*", "Chefe", and "12". At the bottom of the window, there are two buttons: "Cadastrar" and "Voltar". The window has standard OS window controls (minimize, maximize, close) in the title bar.

Label	Value
Nome:	Marcos
Login:	MarcosChefe
Senha:	***
Cargo:	Chefe
Identificação:	12

Figura 3 – Cadastro Entidade do Governo

#### 4.3 Cadastro Diretor

Tela do programa que realiza o cadastro da diretor. Caso a alguma informação do login esteja errada ou exista um o usuário com mesmo login o usuário será avisado.

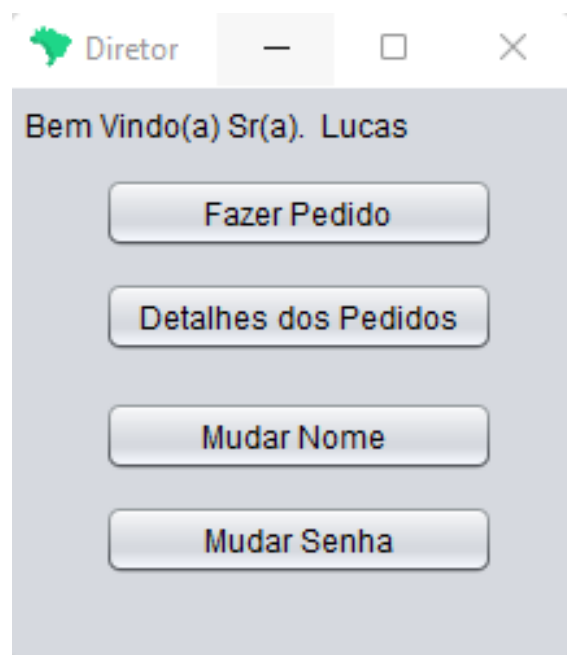


A screenshot of a software window titled "Cadastrar ..." with a green tree icon. The window contains a registration form with the following fields: "Nome:" with the value "Lucas", "Login:" with the value "LucasDiretor", "Senha:" with masked characters "\*\*\*", "CPF:" with the value "05843803135", "Tel:" with the value "(61) 99283-6375", and "Escola:" with a dropdown menu showing "Escola1". At the bottom, there are two buttons: "Cadastrar" and "Voltar".

Figura 4 – Cadastro Diretor

#### 4.4 Tela Inicial do Diretor

Tela inicial do diretor. Dá acesso a outras telas.



A screenshot of a software window titled "Diretor" with a green tree icon. The window displays a welcome message: "Bem Vindo(a) Sr(a). Lucas". Below the message, there are four buttons arranged vertically: "Fazer Pedido", "Detalhes dos Pedidos", "Mudar Nome", and "Mudar Senha".

Figura 5 – Tela Inicial do Diretor

#### 4.5 Tela de pedidos do Diretor

Diretor pode realizar pedidos de 3 tipos, porém apenas 1 de cada tipo.

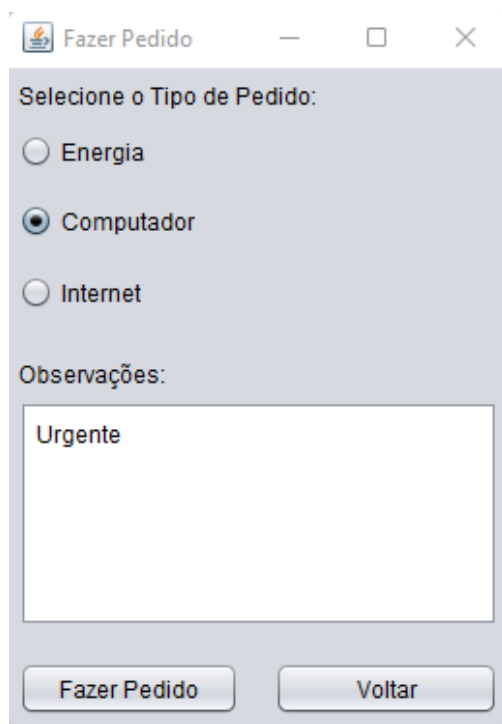
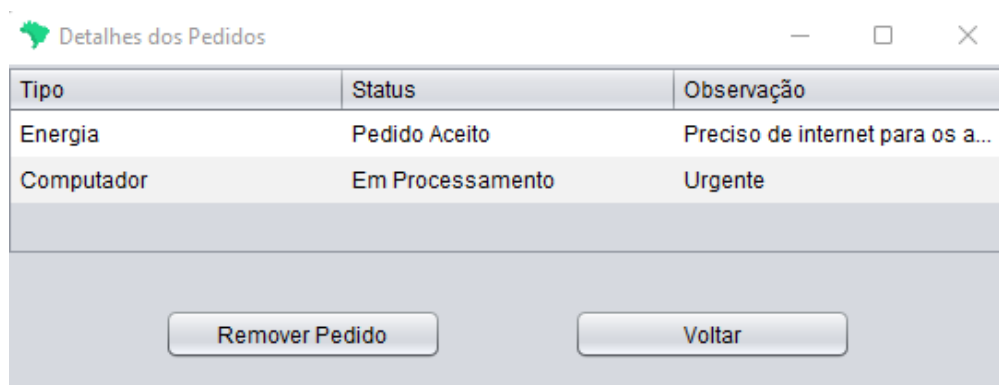


Figura 6 – Tela de pedidos do Diretor

#### 4.6 Tela de detalhes de pedidos do Diretor

Diretor pode ver pedidos feitos e suas observações além de poder excluir.



Tipo	Status	Observação
Energia	Pedido Aceito	Preciso de internet para os a...
Computador	Em Processamento	Urgente

Figura 7 – Tela de detalhes de pedidos do Diretor

#### 4.7 Trocar senha ou nome de usuário

Em telas como essa podem ser feitas as mudanças de nome login e senha dos usuários.

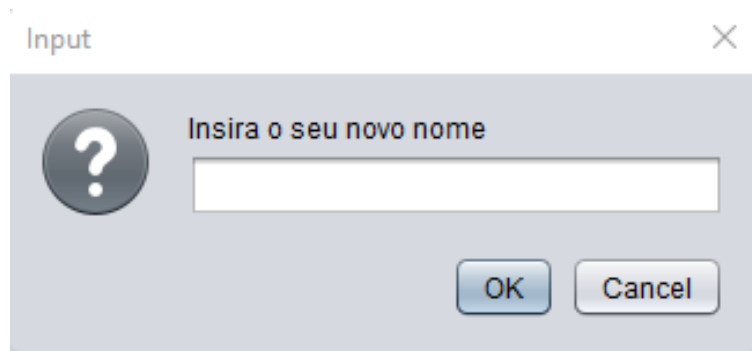


Figura 8 – Trocar senha ou nome de usuário

#### 4.8 Tela inicial da entidade

Tela inicial da entidade, nessa tela podemos acessar as tabelas de chamados e pedidos.



Figura 9 – Tela inicial da entidade

#### 4.9 Tela de pedidos

Nessa tela podemos ver todos os pedidos feitos por todos os diretores e podemos gerenciar criando chamados com esses pedidos.



**Pedidos**

Diretor: Alirio

Escola: Escola2

Tipo: Internet

Aceito: true

Observação: Realizado| Atualização | Chamado completo

Fazer Chamado Voltar

**Selecione o pedido para realizar o chamado:**

Diretor	Escola	Aceito	Tipo	Observação
Lucas	Escola1	true	Energia	Preciso de internet p...
Lucas	Escola1	true	Computador	Urgente  Atualização...
Alirio	Escola2	false	Energia	Help
Alirio	Escola2	true	Computador	Atualização   Pedid...
Alirio	Escola2	true	Internet	Atualização   Pedid...

Figura 10 – Tela de pedidos

#### 4.10 Tela de Chamado de Energia

Nessa tela podemos ver todos os chamados feitos e temos a opção de resolver os chamados, verificando as informações da escola e se é viável ou não realizar esse chamado.

**Chamados de Energia**

Chamado de energia

Resolver Voltar

Diretor	Escola	Numero	Resolvido
Lucas	Escola1	1	true
Alirio	Escola2	1	false

Figura 11 – Tela de Chamado de Energia

#### 4.11 Tela de Chamado de Internet

Nessa tela podemos ver todos os chamados feitos e temos a opção de resolver os chamados, verificando as informações da escola e se é viável ou não realizar esse chamado.

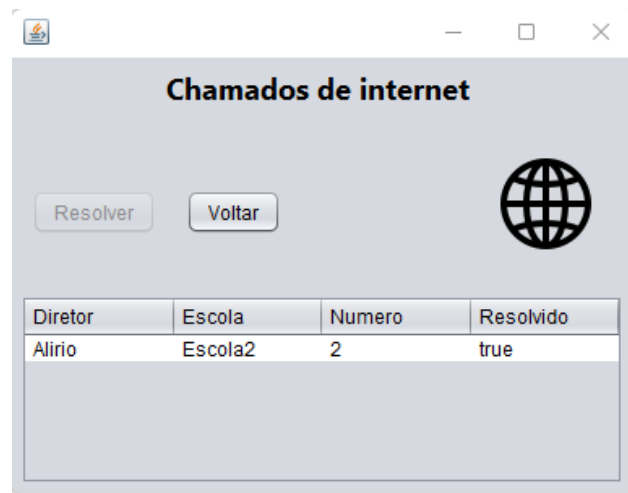


Figura 12 – Tela de Chamado de Internet

#### 4.12 Tela de Chamado de Computador

Nessa tela podemos ver todos os chamados feitos e temos a opção de resolver os chamados, verificando as informações da escola e se é viável ou não realizar esse chamado.

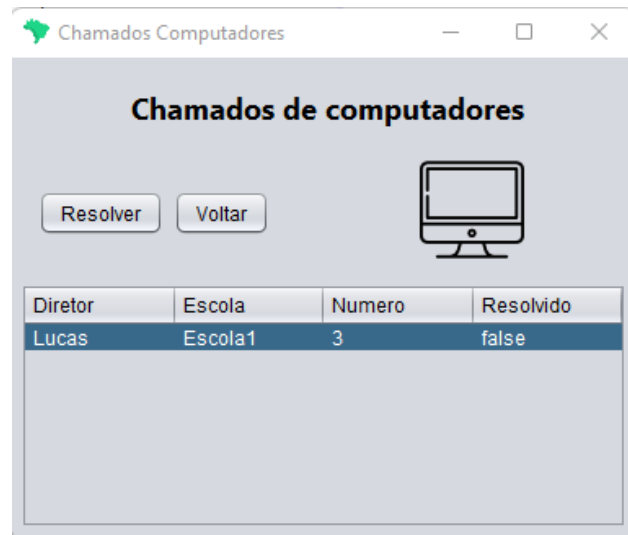


Figura 13 – Tela de Chamado de Computador