

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA
CENTRO DE EDUCAÇÃO SUPERIOR DO ALTO VALE DO
ITAJAÍ**

**CURSO DE ENGENHARIA DE SOFTWARE
PERSISTÊNCIA DE DADOS**

JÉSSICA BERNARDI PETERSEN

GERENCIADOR DE REGISTROS

**Ibirama – SC
2018**

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA
CENTRO DE EDUCAÇÃO SUPERIOR DO ALTO VALE DO
ITAJAÍ**

**CURSO DE ENGENHARIA DE SOFTWARE
PERSISTÊNCIA DE DADOS**

JÉSSICA BERNARDI PETERSEN

GERENCIADOR DE REGISTROS

Trabalho apresentado como exigência para
fechamento das médias finais da matéria de
persistência de dados do curso de Engenharia
de Software da Universidade do Estado de
Santa Catarina.

Prof. Dr. Tiago Luiz Schmitz

**Ibirama – SC
2018**

LISTA DE FIGURAS

Figura 1: Classe TrabalhoPersistencia.....	06
Figura 2: Classe Usuario.....	07
Figura 3: Método main().....	08
Figura 4: Método criaArquivo().....	09
Figura 5: Método adicionaUsuario().....	09
Figura 6: Método cadastralInfoUsuario().....	10
Figura 7: Método verificaCadastrado.....	11
Figura 8: Método hashInicialCodigo.....	11
Figura 9: Método verificaProximosCadastrados.....	12
Figura 10: Método completaString().....	12
Figura 11: Método gravar().....	13
Figura 12: Método verificaOndeGravar().....	14
Figura 13: Método gravaNovoArquivoComProximo().....	14
Figura 14: Método gravaNovoArquivoSemProximo().....	14
Figura 15: Método gravaPosicoesProximas().....	15
Figura 16: Método removerUsuario().....	16
Figura 17: Método procurarPor().....	16
Figura 18: Método removerCodigo().....	17
Figura 19: Método removerCodigoProximo().....	17
Figura 20: Método removerPorNome().....	18
Figura 21: Método ProcuraNome().....	19

Figura 22: Método alterarUsuario()	20
Figura 23: Método procuraCodigo()	20
Figura 24: Método consultar()	21
Figura 25: Método consultarPorCodigo()	22
Figura 26: Método mostraConsulta()	22
Figura 27: Método consultarPorNome()	23
Figura 28: Método procurarPorNome()	23
Figura 29: Método mostrarTudo()	24
Figura 20: Método mostrarPorHash()	25

SUMÁRIO

LISTA DE FIGURAS.....	III
1.INTRODUÇÃO.....	06
2.DETALHAMENTO DAS ATIVIDADES.....	06
2.1. Apresentação da estrutura base.....	06
2.2. Inserir registro.....	09
2.3. Remover registro.....	15
2.4. Alterar registro.....	19
2.5. Consultar registro.....	21
2.6. Mostrar todos registros.....	24
2.7. Mostrar todos registros por hash.....	25
3.CONCLUSÃO.....	26

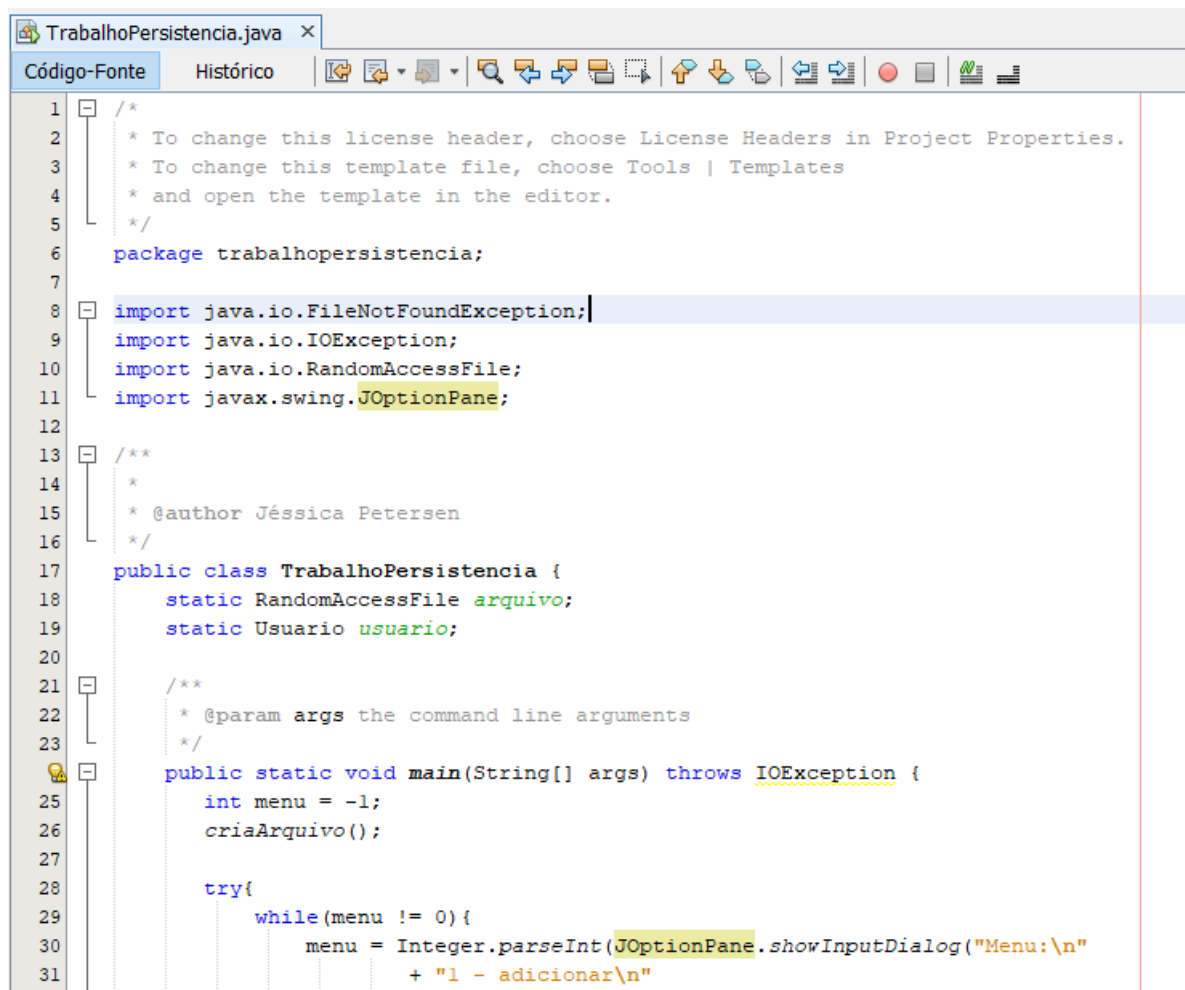
1. INTRODUÇÃO

O presente trabalho visa implementar um gerenciamento de registros em arquivo em java seguindo um hash de base 28. O software permite adicionar, remover, alterar e consultar os registros do usuário, código, nome, qualificação e salário. O atributo chave é o código e é informado pelo usuário. A exclusão do registro é feita de forma lógica.

2. DETALHAMENTO DAS ATIVIDADES

2.1. Apresentação da estrutura base

Foi desenvolvido duas classes: TrabalhoPersistencia (Figura 1), ao qual possui o método main e todos os métodos implementados para o funcionamento do software e a classe Usuario (Figura 2), ao qual possui os atributos código, nome, salário, qualificação e próximo com os getters e setters.

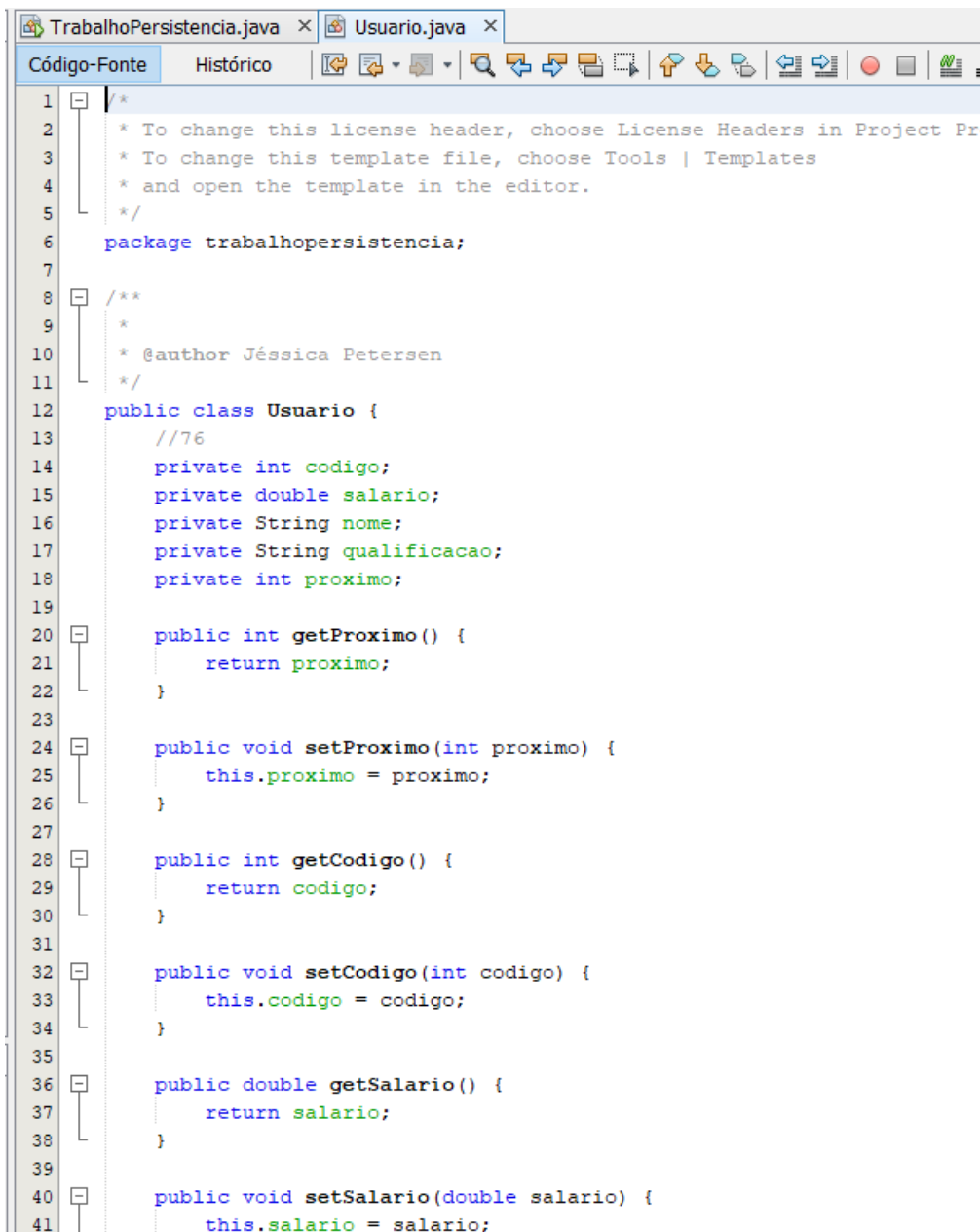


```

1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package trabalhopersistencia;
7
8   import java.io.FileNotFoundException;
9   import java.io.IOException;
10  import java.io.RandomAccessFile;
11  import javax.swing.JOptionPane;
12
13  /**
14   *
15   * @author Jéssica Petersen
16   */
17  public class TrabalhoPersistencia {
18      static RandomAccessFile arquivo;
19      static Usuario usuario;
20
21      /**
22       * @param args the command line arguments
23       */
24      public static void main(String[] args) throws IOException {
25          int menu = -1;
26          criaArquivo();
27
28          try{
29              while(menu != 0){
30                  menu = Integer.parseInt(JOptionPane.showInputDialog("Menu:\n"
31                      + "1 - adicionar\n"

```

Figura 1: Classe TrabalhoPersistencia



```

1  /*
2   * To change this license header, choose License Headers in Project Pr
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package trabalhopersistencia;
7
8   /**
9    *
10   * @author Jéssica Petersen
11   */
12   public class Usuario {
13       //76
14       private int codigo;
15       private double salario;
16       private String nome;
17       private String qualificacao;
18       private int proximo;
19
20       public int getProximo() {
21           return proximo;
22       }
23
24       public void setProximo(int proximo) {
25           this.proximo = proximo;
26       }
27
28       public int getCodigo() {
29           return codigo;
30       }
31
32       public void setCodigo(int codigo) {
33           this.codigo = codigo;
34       }
35
36       public double getSalario() {
37           return salario;
38       }
39
40       public void setSalario(double salario) {
41           this.salario = salario;

```

Figura 2: Classe Usuario

Dentro do método main (Figura 3), é chamado o método criaArquivo() (Figura 4), e então é criado o menu, que possibilita ao usuário escolher a opção desejada: adicionar um usuário ao arquivo, remover, alterar, consultar, mostrar todos os usuários presentes no arquivo e mostrar todos os arquivos de acordo com o hash desejado.

```

public static void main(String[] args) throws IOException {
    int menu = -1;
    criaArquivo();

    try{
        while(menu != 0){
            menu = Integer.parseInt(JOptionPane.showInputDialog("Menu:\n"
                + "1 - adicionar\n"
                + "2 - remover\n"
                + "3 - alterar\n"
                + "4 - consultar\n"
                + "5 - Mostrar todos\n"
                + "6 - Mostrar todos de um determinado hash\n"
                + "0 - sair"));

            switch(menu){
                case 1:
                    adicionaUsuario();
                    break;
                case 2:
                    removerUsuario();
                    break;
                case 3:
                    alterarUsuario();
                    break;
                case 4:
                    consultar();
                    break;
                case 5:
                    mostrarTudo(0, "");
                    break;
                case 6:
                    int hash = Integer.parseInt(JOptionPane.showInputDialog("Digite o hash desejado (de 0 a 27)"));
                    if(hash >= 0 && hash < 28){
                        String mostrar = "Hash: " + hash;
                        mostrar = mostrarPorHash(hash*136, "");
                        JOptionPane.showMessageDialog(null, mostrar);
                    }else{
                        JOptionPane.showMessageDialog(null, "Hash inválido");
                    }
                    break;
            }
        }
    } catch(Exception e){
        System.out.println(e);
    }
}

```

Figura 3: Método main()

O método `criaArquivo()` (Figura 4) é responsável por criar o arquivo csv e completar o arquivo com os 28 hash com registros vazios.

Primeiramente o ponteiro do arquivo é jogado para o início do arquivo (posição 0) para iniciar a gravar os 28 hash vazios. É gravado o código com o valor de -1 para indicar que não há usuário cadastrado, em seguida é gravado 60 chars vazios, sendo 30 para o nome e os próximos 30 para a qualificação, o salário é gravado com o valor double 0 e em seguida o próximo é gravado com -1 para indicar que não possui próximo. O for é o responsável por criar o arquivo com os 28 hash vazios.


```

/**
 * Cria os 28hash vazios
 * @throws FileNotFoundException
 * @throws IOException
 */
public static void criaArquivo() throws FileNotFoundException, IOException{
    arquivo = new RandomAccessFile("trabalhoPersistencia.csv", "rw" );
    arquivo.seek(0);
    for (int i = 0; i < 28; i++) {
        arquivo.writeInt(-1);
        arquivo.writeChars(String.format("%1$60s", ""));
        arquivo.writeDouble(0);
        arquivo.writeInt(-1);
    }
}

```

Figura 4: Método criaArquivo()

2.2. Inserir registro

Ao escolher a opção 1 do menu (Figura 3) – Adicionar um registro, chamará o método adicionaUsuario() (Figura 5). Esse método cria um novo objeto usuário, chama o método cadastraInfoUsuario() (Figura 6) passando esse objeto recém criado e um false ao qual informa que é para cadastrar e não alterar. Em seguida cadastra o atributo próximo como -1 para informar que não tem próximo e chamada o método gravar() (Figura 11) passando o objeto usuário.

```

/**
 * Adiciona um usuário
 * @throws IOException
 */
public static void adicionaUsuario() throws IOException{
    usuario = new Usuario();
    cadastraInfoUsuario(usuario, false);
    usuario.setProximo(-1);
    gravar(usuario);
}

```

Figura 5: Método adicionaUsuario()

No método cadastraInfoUsuario() (Figura 6) irão acontecer as validações e fazer a interação com o usuário, coletando os dados para gravar no registro. Caso seja uma gravação de dados (e não alteração) irá pedir ao usuário qual o código do usuário a ser cadastrado e já salvar no objeto usuário. Logo em seguida chama o método verificaCadastrado() (Figura 7) ao qual fará a validação se o código informado já foi cadastrado anteriormente, caso já tenha sido, informa ao usuário que já foi cadastrado e pede novamente para informar um código.

```

/**
 * Método que cadastra as informacoes no objeto usuário
 * @param usuario
 */
public static void cadastraInfoUsuario(Usuario usuario, boolean alterar) throws IOException{
    boolean certo;
    if(!alterar){
        do{
            usuario.setCodigo(Integer.parseInt(JOptionPane.showInputDialog("Informe o código do usuário")));
            certo = verificarCadastrado(usuario.getCodigo());
            if(certo){
                JOptionPane.showMessageDialog(null, "Código já cadastrado");
            }
        }while(certo);
    }
    certo = false;
    while(!certo){
        String nome = JOptionPane.showInputDialog("Informe o nome do usuário");
        if (nome.length() < 31) {
            if (nome.length() < 30) {
                nome = completaString(nome);
            }
            usuario.setNome(nome);
            certo = true;
        }else{
            JOptionPane.showMessageDialog(null, "Nome do usuário muito grande!");
        }
    }
    usuario.setSalario(Double.parseDouble(JOptionPane.showInputDialog("Informe o salário")));
    certo = false;
    while(!certo){
        String qualificacao = JOptionPane.showInputDialog("Informe a qualificacao do usuário");
        if (qualificacao.length() < 31) {
            if (qualificacao.length() < 30) {
                qualificacao = completaString(qualificacao);
            }
            usuario.setQualificacao(qualificacao);
            certo = true;
        }else{
            JOptionPane.showMessageDialog(null, "Qualificacao muito grande!");
        }
    }
}

```

Figura 6: Método cadastraInfoUsuario()

Em seguida o usuário deverá informar o nome do usuário onde será realizada a validação para verificar se o nome informado não possui mais do que 30 caracteres, se passar, terá que informar outro nome, e se for menor do que 30 chama o método `completaString()` (Figura 10) ao qual vai completar até 30 caracteres com espaços em branco. Para a qualificação é feita a mesma validação e chama o mesmo método para completar a String.

No método `verificaCadastrado()` (Figura 7), é passado o código ao qual está sendo procurado, esse código é passado no método `hashInicialCodigo()` (Figura 8) ao qual retornará a posição do primeiro hash ao qual o ponteiro do arquivo irá começar a leitura do código. Caso o código procurado seja igual ao código no

arquivo, retornará true, indicando que o código já foi cadastrado, caso não seja, é chamado o método verificaProximosCadastrados() (Figura 9) e ao final retorna true (achou) ou false (não foi achado).

```
/**
 * Verifica se o código que está sendo cadastrado já foi cadastrado anteriormente
 * @param codigoProcurado int
 * @return bool
 */
public static boolean verificarCadastrado(int codigoProcurado) throws IOException{
    boolean achou = false;
    int posicao = hashInicialCodigo(codigoProcurado);
    arquivo.seek(posicao);
    int codigoLeitura = arquivo.readInt();
    if (codigoLeitura == codigoProcurado) {
        return true;
    }else{
        achou = verificaProximosCadastrados(posicao+132, codigoProcurado, achou);
    }

    return achou;
}
```

Figura 7: Método verificaCadastrado()

Ao informar o código no método hashInicialCodigo() (Figura 8), é realizado o módulo por 28 para verificar em qual hash aquele código pertence e retornará a posição inicial no arquivo do primeiro hash.

```
/**
 * Retorna a posicao inicial(hash inicial) que o código vai pertencer
 * @param codigo
 * @return posicao
 * @throws IOException
 */
public static int hashInicialCodigo(int codigo) throws IOException{
    int hash = codigo%28;
    return (hash*136);
}
```

Figura 8: Método hashInicialCodigo()

O método verificaProximosCadastrados() (Figura 9) recebe como parâmetros a posição ao qual o arquivo deve ler para procurar a posição do próximo código, o código procurado e um booleano se encontrou ou não o código procurado. Após fazer a leitura do código, é verificado se ele é -1, caso for, ele retorna false pois não existe mais nenhum arquivo após o recém lido. Se for diferente de -1, o ponteiro do arquivo é apontado para a posição do próximo registro e é feito a leitura do código.

Caso esse código recém (codigoLeituro) lido seja igual a -1 (não existe registro preenchido), verifica se tem próximo, se não existir (igual a -1) retorna false, se existir, faz a recursividade do método para ir procurando nas posições seguintes. Caso o

código (codigoLeituro) for diferente de -1, verifica se o código lido é igual ao código procurado, se for, retorna true indicando que foi achado, se não for igual, faz a recursividade do método procurando nas posições seguintes.

```
/**
 * Faz a leitura para verificar se o código que está sendo cadastrado já foi cadastrado anteriormente;
 * @param proximaLeitura
 * @param codigoProcurado
 * @return boolean
 * @throws IOException
 */
public static boolean verificaProximosCadastrados(int proximaLeitura, int codigoProcurado, boolean achou) throws IOException {
    arquivo.seek(proximaLeitura);
    int proximaPosicao = arquivo.readInt();
    if (proximaPosicao != -1) {
        arquivo.seek(proximaPosicao);
        int codigoLeituro = arquivo.readInt();
        if (codigoLeituro == -1) {
            arquivo.seek(proximaPosicao+132);
            int proxP = arquivo.readInt();
            if (proxP == -1) {
                return false;
            } else {
                achou = verificaProximosCadastrados(proximaPosicao+132, codigoProcurado, achou);
            }
        } else {
            if (codigoLeituro != codigoProcurado) {
                achou = verificaProximosCadastrados(proximaPosicao+132, codigoProcurado, achou);
            } else {
                return true;
            }
        }
    }
    return achou;
}
```

Figura 9: Método verificaProximosCadastrados()

O método completaString (Figura 10) recebe como parâmetro a string a ser completada e com o for, completa até a string possuir 30 caracteres e a retorna.

```
/**
 * Completa a string para ficar com 30 char
 * @param completar
 * @return
 */
public static String completaString(String completar) {
    for (int i = completar.length(); i < 30; i++) {
        completar += " ";
    }
    return completar;
}
```

Figura 10: Método completaString()

O método gravar() recebe o objeto usuario como parâmetro para gravar os dados desse objeto no arquivo. Inicialmente é encontrado a posição do primeiro hash de acordo com o seu código através do método hashInicialCodigo() (Figura 8). O ponteiro do arquivo é apontado para aquela posição onde será verificado se aquele arquivo

possui ou não algum registro gravado. Caso não tenha (código == -1), é chamado o método `verificaOndeGravar()` (Figura 12), caso o hash já possua um registro, é verificado se ele aponta para um próximo, caso não aponte (`proximoCodigo == -1`), é gravado a ultima posição do arquivo no próximo desse ultimo código e é chamado o método `gravaPosicoesProximas()` (Figura 15) passando a ultima posição do arquivo, caso exista próximo (`proximoCodigo != -1`), é chamado o método `gravaPosicoesProximas()` e é passado o código do próximo ao qual ele aponta.

```
/**
 * Realiza o processo de inserção
 * @param usuario
 * @throws FileNotFoundException
 * @throws IOException
 */
public static void gravar(Usuario usuario) throws FileNotFoundException, IOException{
    int posicaoInicialArquivo = hashInicialCodigo(usuario.getCodigo());
    arquivo.seek(posicaoInicialArquivo);
    int codigo = arquivo.readInt();
    if (codigo == -1) {
        verificaOndeGravar(posicaoInicialArquivo + 132, posicaoInicialArquivo);
    }else{ // se já possui o primeiro hash -- procurar nas proximas posições
        arquivo.seek(posicaoInicialArquivo+132);
        int proximoCodigo = arquivo.readInt();
        if(proximoCodigo == -1){
            arquivo.seek(posicaoInicialArquivo+132);
            arquivo.writeInt((int)arquivo.length());
            gravarPosicoesProximas((int)arquivo.length());
        }else{
            gravarPosicoesProximas(proximoCodigo);
        }
    }
}
```

Figura 11: Método `gravar()`

No método `verificaOndeGravar()` (Figura 12), a posição em que o próximo código se encontra e a posição inicial do arquivo são passadas como parâmetros. O ponteiro do arquivo é direcionado para a posição informada onde é realizada a leitura da posição de onde se encontra o próximo registro para então o ponteiro ser direcionado para esta posição. Se o código lido for igual a -1, quer dizer que não havia nada antes, então chamara o método `gravaNovoArquivoComProximo()` (Figura 13) para gravar as informações salvando a posição próximo, caso existia um registro anteriormente, chama o método `gravaNovoArquivoSemProximo()` (Figura 14) para salvar sem sobrescrever a posição próximo.

```

/**
 * Verifica se na posicao hash já havia um arquivo anteriormente, se ja tinha, grava sem sobrescrever o 'próximo'
 * @param posicaoProximoCodigo
 * @param posicaoInicialArquivo
 * @throws IOException
 */
public static void verificaOndeGravar(int posicaoProximoCodigo, int posicaoInicialArquivo) throws IOException{
    arquivo.seek(posicaoProximoCodigo);
    int proximoCodigo = arquivo.readInt();
    arquivo.seek(posicaoInicialArquivo);
    if (proximoCodigo == -1) { // nao existia nada
        gravaNovoArquivoComProximo();
    }else{
        gravaNovoArquivoSemProximo();
    }
}
}

```

Figura 12: Método verificaOndeGravar()

```

/**
 * Grava um novo arquivo em uma posição que não havia um usuario salvo anteriormente, e que consequentemente possui um próximo (fica no meio)
 * @throws IOException
 */
public static void gravaNovoArquivoComProximo() throws IOException{
    arquivo.writeInt(usuario.getCodigo());
    arquivo.writeChars(usuario.getNome());
    arquivo.writeChars(usuario.getQualificacao());
    arquivo.writeDouble(usuario.getSalario());
    arquivo.writeInt(usuario.getProximo());
}

```

Figura 13: Método gravaNovoArquivoComProximo()

```

/**
 * Grava um novo arquivo em uma posição que havia um usuário salvo anteriormente, porém ele era o último (não havia próximo)
 * @throws IOException
 */
public static void gravaNovoArquivoSemProximo() throws IOException{
    arquivo.writeInt(usuario.getCodigo());
    arquivo.writeChars(usuario.getNome());
    arquivo.writeChars(usuario.getQualificacao());
    arquivo.writeDouble(usuario.getSalario());
}

```

Figura 14: Método gravaNovoArquivoSemProximo()

O método gravaPosicoesProximas() (Figura 15) recebe a posição atual como parametro e verifica se essa posição passada é igual ao tamanho do arquivo, caso for, grava as informações do usuário no final do arquivo através do método gravaNovoArquivoComProximo() (Figura 13), caso contrário, salva o código na variável posicaoProximo para fazer a verificação se tem registro ou não e já salva o posição do próximo na variável proximaPosicao para verificar se possui um arquivo após esse ou não.

Caso não tenha nada salvo na posição atual (PosicaoProximo == -1), é verificado se já havia algo salvo anteriormente ou não, caso não existia, salva através do método gravaNovoArquivoComProximo() (Figura 13) se não grava com o método gravaNovoArquivoSemProximo() (Figura 14). Se Possui algo salvo na posição atual,

verifica se a próxima posição tem alguma coisa, se não tiver, salva no final do arquivo, se existe algo, faz sua recursiva até encontrar uma posição para salvar.

```
/**
 * Verifica onde tem uma posição 'vazia' para gravar
 * @param posicaoAtual
 * @throws IOException
 */
public static void gravarPosicoesProximas(int posicaoAtual) throws IOException{
    if (posicaoAtual == arquivo.length()) {
        arquivo.seek(posicaoAtual);
        gravaNovoArquivoComProximo();
    }else{
        arquivo.seek(posicaoAtual);
        int posicaoProximo = arquivo.readInt();
        arquivo.seek(posicaoAtual+132);
        int proximaPosicao = arquivo.readInt();
        //verifica se a posição atual tem ou não registro, se não tiver, verificar se já tinha registro anteriormente
        if(posicaoProximo == -1){
            arquivo.seek(posicaoAtual);
            if(proximaPosicao == -1){
                gravaNovoArquivoComProximo();
            }else{
                gravaNovoArquivoSemProximo();
            }
        }else{
            if(proximaPosicao == -1){
                arquivo.seek(posicaoAtual+132);
                arquivo.writeInt((int)arquivo.length());
                gravarPosicoesProximas((int)arquivo.length());
            }else{
                gravarPosicoesProximas(proximaPosicao);
            }
        }
    }
}
```

Figura 15: Método gravaPosicoesProximas()

2.3. Remover registro

É chamado o método removerUsuario() (Figura 16) quando a opção desejada for a número 2 – remover, conforme o menu (Figura 3). Esse método chama o método procurarPor() (Figura 17), que pergunta ao usuário se ele deseja remover o usuário por código ou por nome. Caso ele opte pela opção por código, é chamado o método removerCodigo() (Figura 18) e se optar pela remoção por nome, é chamado o método removerPorNome() (Figura 20). Caso o usuário tenha apertado na opção de remoção sem querer, ele poderá optar por voltar ao menu principal digitando 0.

```

/**
 * Remove o usuário de acordo com desejado (Código ou nome)
 * @throws IOException
 */
public static void removerUsuario() throws IOException{
    int menu;
    do{
        menu = procurarPor();
        switch(menu){
            case 1:
                removerCodigo();
                break;
            case 2:
                removerPorNome();
                break;
        }
    }while(menu !=0);
}

```

Figura 16: Método removerUsuario()

```

/**
 * Menu
 * @return
 */
public static int procurarPor(){
    int menu = Integer.parseInt(JOptionPane.showInputDialog("Procurar por:\n"
        + "1 - código\n"
        + "2 - nome\n"
        + "0 - voltar ao menu principal"));
    return menu;
}

```

Figura 17: Método procurarPor()

No método removerCodigo() (Figura 18), é solicitado ao usuário o código que ele deseja remover e é realizado a verificação se existe o código digitado pelo usuário gravado no arquivo através do método verificarCadastrado (Figura 7), caso não exista, uma mensagem é exibida, se existir, o ponteiro do arquivo é colocado no hash inicial através do método hashInicialCodigo() (Figura 8) e é verificado se o código presente na posição do hash inicial é o código procurado, caso seja, é sobrescrito em cima da posição o código -1 para indicar que não tem mais registro ali bem como colocado caracteres em branco em cima do que existia anteriormente.

Caso não seja o código buscado, é chamado o método removerCodigoProximos() (Figura 19), passando como parâmetro a posição no arquivo onde vai estar a posição do próximo código e o código buscado.


```

/**
 * Remove por código
 * @throws IOException
 */
public static void removerCodigo() throws IOException{
    int codigo = Integer.parseInt(JOptionPane.showInputDialog("Informe o código para remoção"));
    if(!verificarCadastrado(codigo)){
        JOptionPane.showMessageDialog(null, "Código não cadastrado");
    }else{
        int posicao = hashInicialCodigo(codigo);
        arquivo.seek(posicao);
        int codigoCadastrado = arquivo.readInt();
        if (codigoCadastrado == codigo) {
            arquivo.seek(posicao);
            arquivo.writeInt(-1);
            arquivo.writeChars(String.format("%1$30s", ""));
            arquivo.writeChars(String.format("%1$30s", ""));
            arquivo.writeDouble(0);
            // tem ou não próximo -- escrever -1 ou deixar como
        }else{
            removerCodigoProximos(posicao+132, codigo);
        }
    }
}

```

Figura 18: Método removerCodigo()

O método removerCodigoProximos() (Figura 19) faz a leitura na posição passada e salva na variável posicaoProximo, é verificado se possui algum código na leitura do próximo, caso exista, o ponteiro do arquivo é colocado na posição e é feita a leitura do código, caso seja igual ao buscado, é feita a sobrescrita do registro para indicar que foi excluído, caso ainda não seja o código buscado, é feito a recursividade do método até encontrar o código buscado.

```

/**
 * Procura próximos para remover
 * @param proxPosicao
 * @param codigoProcurado
 * @throws IOException
 */
public static void removerCodigoProximos(int proxPosicao, int codigoProcurado) throws IOException{
    arquivo.seek(proxPosicao);
    int posicaoProximo = arquivo.readInt();
    if(posicaoProximo != -1){
        arquivo.seek(posicaoProximo);
        int codicoProximo = arquivo.readInt();
        if (codicoProximo == codigoProcurado) {
            arquivo.seek(posicaoProximo);
            arquivo.writeInt(-1);
            arquivo.writeChars(String.format("%1$30s", ""));
            arquivo.writeChars(String.format("%1$30s", ""));
            arquivo.writeDouble(0);
            JOptionPane.showMessageDialog(null, "Removido com sucesso!");
        }else{
            removerCodigoProximos(posicaoProximo+132, codigoProcurado);
        }
    }
}

```

Figura 19: Método removerCodigoProximos()

```

/**
 * Método que remove o usuário por nome
 * @throws IOException
 */
public static void removerPorNome() throws IOException{
    arquivo.seek(0);
    String nome;
    boolean certo = false;
    do{
        nome = JOptionPane.showInputDialog("Informe o nome do usuário");
        if(nome.length() < 31 ){
            certo = true;
            if(nome.length() < 30){
                for (int i = nome.length(); i < 30; i++) {
                    nome += " ";
                }
            }
        }else{
            JOptionPane.showMessageDialog(null, "Nome informado muito grande!");
        }
    }while(!certo);
    procuraNome(0+4, nome);
}

```

Figura 20: Método removerPorNome()

O método `removerPorNome()` (Figura 20) aponta o ponteiro do arquivo para a posição 0 (inicial), pois será necessário verificar em todos os nomes presentes no arquivo para fazer a remoção. Após o usuário informar o nome, é feita a verificação se o nome apresenta menos ou até 30 caracteres. Após a verificação de que é menor, a string é completada com espaços em branco e o método `procuraNome()` (Figura 21) é chamado.

No método `procuraNome()` (Figura 21), é verificado se a posição não é maior que o arquivo, caso for, termina o processo e aparece a mensagem, se for menor, faz a leitura do código e verifica se tem registro, se não tem, faz a recursiva para verificar a próxima posição, se tiver registro, verifica se os caracteres do nome batem com os caracteres da palavra buscada, se forem iguais, é feito a “remoção” dos atributos registrados anteriormente para vazio e ao final é feito a recursiva verificar as próximas posições.

```

/**
 * Procura o usuário por nome e já remove
 * @param posicaoBusca
 * @param nomeBuscado
 * @throws IOException
 */
public static void procuraNome(int posicaoBusca, String nomeBuscado, String msg) throws IOException{
    if(posicaoBusca < arquivo.length()){
        arquivo.seek(posicaoBusca);
        int codigo = arquivo.readInt();
        if( codigo != -1){
            boolean achou = true;
            for (int i = 0; i < 30; i++) {
                char a = arquivo.readChar();
                if (a != nomeBuscado.charAt(i)) {
                    achou = false;
                    break;
                }
            }
            int posicao = posicaoBusca + 136;
            if(achou){
                arquivo.seek(posicaoBusca);
                arquivo.writeInt(-1);
                arquivo.writeChars(String.format("%1$30s", ""));
                arquivo.writeChars(String.format("%1$30s", ""));
                arquivo.writeDouble(0);
                msg = "Removido com sucesso!";
            }
            procuraNome(posicao, nomeBuscado, msg);
        }else{
            int posicao = (int) arquivo.getFilePointer();
            procuraNome(posicao+132, nomeBuscado, msg);
        }
    }else{
        if(msg.length() < 2){
            JOptionPane.showMessageDialog(null, "Não foi encontrado");
        }else{
            JOptionPane.showMessageDialog(null, msg);
        }
    }
}

```

Figura 21: Método procuraNome()

2.4. Alterar registro

Ao usuário escolher a opção 3 – alterar do menu principal (Figura 3), é chamado o método `alterarUsuario()` (Figura 22). Como o código é a chave primaria ela não pode ser removida, assim o usuário só poderá alterar as outras informações do registro. Inicialmente o usuário deverá informar o código do registro que deseja alterar. Através do método `verificarCadastrado()` (Figura 7), é verificado se o código informado está presente no arquivo, caso esteja, o método `cadastraInfoUsuario()` (Figura 6) é chamado para cadastrar as novas informações no objeto usuário.

Após o cadastramento das informações no objeto, é verificado em qual hash o código alterado se encontra através do método `hashInicialCodigo()` (Figura 8) e então

é chamado o método procuraCodigo() (Figura 23) para retornar em qual posição o ponteiro do arquivo deve apontar para fazer a alteração, gravando por cima as novas informações digitadas pelo usuário.

```
/**
 * Altera o usuário
 * @throws IOException
 */
public static void alterarUsuario() throws IOException{
    int codigo = Integer.parseInt(JOptionPane.showInputDialog("Informe o código que deseja alterar"));
    if(verificarCadastrado(codigo)){
        cadastraInfoUsuario(usuario, true);
        int posicaoHash = hashInicialCodigo(codigo);
        int posicao = procuraCodigo(posicaoHash, codigo);
        arquivo.seek(posicao);
        arquivo.writeInt(usuario.getCodigo());
        arquivo.writeChars(usuario.getNome());
        arquivo.writeChars(usuario.getQualificacao());
        arquivo.writeDouble(usuario.getSalario());
    }else{
        JOptionPane.showMessageDialog(null, "Código não cadastrado");
    }
}
```

Figura 22: Método alterarUsuario()

No método procuraCodigo() (Figura 23), o ponteiro do arquivo é colocado na posição passada como parâmetro e é feita a leitura do código, caso seja igual ao procurado, é retornado a posição, caso não seja, o ponteiro é apontado para a posição onde se encontra o próximo e é feita a leitura, salvando a posição na variável proximaPosicaoProcura, onde então é passada como parâmetro no mesmo método para fazer a recursividade até então encontrar a posição do código buscado.

```

/**
 * Procura por código e retorna o a posição do registro no arquivo
 * @param posicaoProcura
 * @param codigoProcurado
 * @return
 * @throws IOException
 */
public static int procuraCodigo(int posicaoProcura, int codigoProcurado) throws IOException{
    arquivo.seek(posicaoProcura);
    int codigoAchado = arquivo.readInt();
    if(codigoAchado == codigoProcurado){
        return posicaoProcura;
    }else{
        arquivo.seek(posicaoProcura+132);
        int proximaPosicaoProcura = arquivo.readInt();
        posicaoProcura = procuraCodigo(proximaPosicaoProcura, codigoProcurado);
    }
    return posicaoProcura;
}

```

Figura 23: Método procuraCodigo()

2.5. Consultar registro

Ao escolher a opção 4 – Consultar do menu principal, é chamado o método consultar() (Figura 24). O método consultar chama o método procurarPor() (Figura 17) onde o usuário vai escolher entre as opções consultar por código ou nome.

Ao consultar por código, é chamado o método consultarPorCodigo() (Figura 25), caso escolha a opção consultar por nome, é chamado o método consultarPorNome() (Figura XX).

```

/**
 * Mostra os dados do usuário buscado
 * @throws IOException
 */
public static void consultar() throws IOException{
    int menu;
    do{
        menu = procurarPor();
        switch(menu){
            case 1: // código
                consultarPorCodigo();
                break;
            case 2: // nome
                consultarPorNome();
                break;
        }
    }while(menu != 0);
}

```

Figura 24: Método consultar()

No método consultarPorCodigo (Figura 25) o usuário informará o código desejado, onde será verificado se o código já foi cadastrado através do método verificarCadastrado() (Figura 7). Ao verificar que já foi cadastrado, é verificado em qual hash o código pertence através do método hashInicialCodigo (Figura 8) e então é chamado o método procuraCodigo() (Figura 23) onde retornará a posição onde o registro se encontra. Essa posição é passada como parâmetro para o método mostraConsulta() (Figura 26).

O método mostraConsulta (Figura 26) apontará o ponteiro do arquivo para a posição passada e fará a leitura de cada atributo para ser exibido na tela para o usuário.

```

/**
 * Busca os dados do usuário de acordo com o código desejado
 * @throws IOException
 */
public static void consultarPorCodigo() throws IOException{
    int codigo = Integer.parseInt(JOptionPane.showInputDialog("Digite o código para consultar"));
    if(verificarCadastrado(codigo)){
        int hash = hashInicialCodigo(codigo);
        int posicao = procuraCodigo(hash, codigo);
        mostraConsulta(posicao);
    }else{
        JOptionPane.showMessageDialog(null, "Código não cadastrado");
    }
}

```

Figura 25: Método consultarPorCodigo()

```

/**
 * Mostra os dados do usuário de acordo com a posicao informada
 * @param posicao
 * @throws IOException
 */
public static void mostraConsulta(int posicao) throws IOException{
    arquivo.seek(posicao);
    String resultado = "Código - Nome: " + arquivo.readInt() + " - ";
    String nome = "";
    String qualificacao = "";
    for (int i = 0; i < 30; i++) {
        nome += arquivo.readChar();
    }
    for (int i = 0; i < 30; i++) {
        qualificacao += arquivo.readChar();
    }
    resultado += nome + "\n Qualificação: " + qualificacao + "\n Salário: " + arquivo.readDouble();
    JOptionPane.showMessageDialog(null, resultado);
}

```

Figura 26: Método mostraConsulta()

Caso o usuário queira consultar por nome, o método consultarPorNome() (Figura 27) é chamado. Onde pedirá ao usuário para informar o nome desejado, que será passado como parâmetro para o método procuraPorNome() (Figura 28) que retornará a posição do arquivo. Caso tenha retornado a posição -1, o nome não foi cadastrado, retornando a mensagem, caso tenha achado, o método mostraConsulta() (Figura 26) é chamado e é passado a posição do registro.

O método procuraPorNome() (Figura 28) vai verificar se o arquivo já não está no final, caso esteja retornará que não foi encontrado o nome procurado. Caso não esteja no final, recursivamente vai verificando nome por nome, em cada registro, se o nome encontrado no arquivo é o buscado. Ao encontrar, a posição é salva na variável posicaoAchada para ser retornada.

```

/**
 * Consulta Por nome e ja mostra
 * @throws IOException
 */
public static void consultarPorNome() throws IOException{
    String palavra = JOptionPane.showInputDialog("Digite o nome para consultar");
    int posicao = procuraPorNome(palavra, 0);
    if(posicao != -1){
        mostraConsulta(posicao);
    }else{
        JOptionPane.showMessageDialog(null, "Nome não encontrado");
    }
}
}

```

Figura 27: Método consultarPor Nome()

```

/**
 * Procura por nome e retorna a posição do registro no arquivo
 * @param nomeBuscado
 * @param posicaoBusca
 * @return
 * @throws IOException
 */
public static int procuraPorNome(String nomeBuscado, int posicaoBusca) throws IOException{
    int posicaoAchado = -1;
    if(posicaoBusca < arquivo.length()){
        arquivo.seek(posicaoBusca);
        boolean achou = false;
        if(arquivo.readInt() != -1){
            String nomeAchado="";
            for (int i = 0; i < 30; i++) {
                char a = arquivo.readChar();
                if(a != ' '){
                    nomeAchado += a;
                }
            }
            achou = nomeBuscado.equals(nomeAchado);
        }

        if(achou){
            posicaoAchado = posicaoBusca;
            return posicaoAchado;
        }else{
            posicaoAchado = procuraPorNome(nomeBuscado, posicaoBusca+136);
        }
    }
    return posicaoAchado;
}
}

```

Figura 28: Método procuraPorNome()

2.6. Mostrar todos registros

Se o usuário optar pela opção 5 do menu principal – Mostrar tudo, o método `mostrarTudo()` (Figura 29) é chamado. Esse método vai ler todas as posições do registro e ir concatenando numa string que será exibida ao final da leitura de todos os registros.

Esse método é recursivo para ir fazendo a leitura de todos os registros.

```
/**
 * Mostra todos os registros já cadastrados
 * @param posicao
 * @param mostrar
 * @throws IOException
 */
public static void mostrarTudo(int posicao, String mostrar) throws IOException{
    if(posicao < arquivo.length()){
        arquivo.seek(posicao);
        int ponteiro = (int) arquivo.getFilePointer();
        int codigo = arquivo.readInt();
        if(codigo != -1){
            String nome = "";
            String qualificacao = "";
            if(codigo != -1){
                for (int i = 0; i < 30; i++) {
                    char a = arquivo.readChar();
                    if( a != ' '){
                        nome += a;
                    }
                }
                for (int i = 0; i < 30; i++) {
                    char a = arquivo.readChar();
                    if( a != ' '){
                        qualificacao += a;
                    }
                }
                double salario = arquivo.readDouble();
                int proximo = arquivo.readInt();
                mostrar += "Início do Arquivo: "+ ponteiro+"\n"+
                    "Código: " +codigo +"\n"+
                    "Nome: "+ nome+"\n"+
                    "Qualificação: " + qualificacao + "\n"+
                    "Salário: "+ salario+"\n"+
                    "Aponta para a posição do Arquivo: "+proximo+"\n\n";
            }
        }
        mostrarTudo(posicao+136, mostrar);
    }
    else{
        JOptionPane.showMessageDialog(null, mostrar+" Tamanho Total do Arquivo: "+ arquivo.length());
    }
}
```

Figura 29: Método `MostrarTudo()`

2.7. Mostrar todos registros por hash

A última opção é a 6, onde o usuário irá informar o hash em que quer verificar os registros e é feita a validação, verificando se digitou um hash valido, e então o método mostrarPorHash (Figura 30).

Nesse método, a string também vai sendo concatenada para ser exibida ao final, porém é verificado se na posição do próximo, existe algum próximo, e o método vai sendo chamado recursivamente até aparecer um próximo código -1, onde indica o final dos arquivos daquele hash.

```
/**
 * Mostra todos os registros por hash
 * @param posicao -- posicao no arquivo
 * @param mostrar -- string concatenada
 * @return String
 * @throws IOException
 */
public static String mostrarPorHash(int posicao, String mostrar) throws IOException {
    int proximo = -1;
    arquivo.seek(posicao);
    int codigo = arquivo.readInt();
    if (codigo != -1) {
        mostrar += "\n Código: " + codigo;
        String nome = "";
        String qualificacao = "";
        if (codigo != -1) {
            for (int i = 0; i < 30; i++) {
                char a = arquivo.readChar();
                if (a != ' '){
                    nome += a;
                }
            }
            for (int i = 0; i < 30; i++) {
                char a = arquivo.readChar();
                if (a != ' '){
                    qualificacao += a;
                }
            }
            mostrar += "\nNome: " + nome + "\nQualificação: " + qualificacao + "\nSalário: " + arquivo.readDouble() + "\n\n";
            proximo = arquivo.readInt();
        }
    }
    if (proximo == -1) {
        return mostrar;
    } else {
        return mostrarPorHash(proximo, mostrar);
    }
}
```

Figura 30: Método mostraPorHash()

3. CONCLUSÃO

Fazer o gerenciador de registros de arquivo foi trabalhoso porém com dedicação e lógica se tornou um trabalho interessante de se desenvolver.