```java
public class HumanTree {

    //Campos de estrutura
    private HumanTree rightChild, leftChild, parent;
    private static final int MAX_NAME_RESULTS = 40;

    //Campos de informacao
    private String name, cpf, adress;
    private int age;
    private char sex;
    public HumanTree () {
    }
    public HumanTree ( String n, String c, String a, char sexo, int id ) {
            name = n; cpf = c; adress = a; sex = sexo; age = id;
    }

    //Metodos de tratamento de informacoes
    /** Retorna o campo nome */
    public String getName () { return name; }
    /** Retorna o campo Endereco */
    public String getAdress () { return adress; }
    /** Retorna o campo cpf */
    public String getCPF () { return cpf; }
    /** Retorna o campo sexo */
    public char getSex () { return sex; }
    /** Retorna o campo idade */
    public int getAge () { return age; }
    /** Seta o campo nome */
    public void setName (String n) { name = n; }
    /** Seta o campo endereco */
    public void setAdress (String add) { adress = add; }
    /** Seta o campo cpf */
    public void setCPF (String c) { cpf = c; }
    /** Seta o campo sexo */
    public void setSex (char sexo) { sex = sexo; }
    /** Seta o campo idade */
    public void setAge (int ag) { age = ag; }
    /** Copia as informacoes de um nodo para este nodo */
    public void transferInfoFrom( HumanTree nodo ) {
            this.name = nodo.name;
            this.cpf = nodo.cpf;
            this.adress = nodo.adress;
            this.age = nodo.age;
            this.sex = nodo.sex;
    }
    /** Imprime os dados contidos no nodo */
    public void imprime () {
            if ( !name.equals("0") ) { //Impede que seja impresso o nodo "dummy"

System.out.println("\n############################################################");
                    System.out.println("Nome: "+name);
                    System.out.println("Endereco: "+adress);
                    System.out.println("Sexo: "+sex);
                    System.out.println("Idade: "+age);
                    System.out.println("CPF: "+cpf);

System.out.println("############################################################\n");
            }
    }
    //Metodos de arvore
    /** Pesquisa um CPF na arvore e retorna o nodo correspondente */
    HumanTree pesquisaCPF ( String CPF ) {
            HumanTree raiz = this;
            while ( raiz != null ) {
                    if (raiz.cpf.compareToIgnoreCase(CPF) == 0) return raiz;
                    else if (raiz.cpf.compareToIgnoreCase(CPF) < 0) raiz = raiz.rightChild;
                    else raiz = raiz.leftChild;
            } return null;
    }
    /** Pesquisa um nome na arvore e retorna um vetor de resultados */
    HumanTree[] pesquisaNome( String Nome ) {

            HumanTree[] results = new HumanTree [MAX_NAME_RESULTS];
            int itera = 0;
            HumanTree raiz = this;

```

```
 75:             while ( raiz != null ) {
 76:                     if (raiz.name.compareToIgnoreCase(Nome) == 0) {
 77:                             results[itera] = raiz;
 78:                             itera++;
 79:                             raiz = raiz.rightChild;
 80:                     }
 81:                     else if (raiz.name.compareToIgnoreCase(Nome) < 0) raiz = raiz.rightChild;
 82:                     else raiz = raiz.leftChild;
 83:             } return results;
 84:     }
 85:     /** Insere um nodo na arvore usando o CPF como chave */
 86:     void insertByCPF (HumanTree nodo) {
 87:             HumanTree folha = this;
 88:             while ( true ) {
 89:                     if ( folha.cpf.compareToIgnoreCase(nodo.cpf) > 0) {
 90:                             if (folha.leftChild == null) {
 91:                                     folha.leftChild = nodo;
 92:                                     nodo.parent = folha;
 93:                                     break;
 94:                             } else folha = folha.leftChild;
 95:                     } else {
 96:                             if (folha.rightChild == null) {
 97:                                     folha.rightChild = nodo;
 98:                                     nodo.parent = folha;
 99:                                     break;
100:                             } else folha = folha.rightChild;
101:                     }
102:             }
103:     }
104:     /** Insere um nodo na arvore usando o nome com chave */
105:     void insertByNome (HumanTree nodo) {
106:             HumanTree folha = this;
107:             while ( true ) {
108:                     if ( folha.name.compareToIgnoreCase(nodo.name) > 0) {
109:                             if (folha.leftChild == null) {
110:                                     folha.leftChild = nodo;
111:                                     nodo.parent = folha;
112:                                     break;
113:                             } else folha = folha.leftChild;
114:                     } else {
115:                             if (folha.rightChild == null) {
116:                                     folha.rightChild = nodo;
117:                                     nodo.parent = folha;
118:                                     break;
119:                             } else folha = folha.rightChild;
120:                     }
121:             }
122:     }
123:     /** Remove um nodo da arvore usando o CPF como chave */
124:     void remove( HumanTree nodo ) {
125:             //Se tiver dois filhos remove o antecessor
126:             if (nodo.leftChild != null && nodo.rightChild != null) nodo.removeAntecessor();
127:             //Se nao tiver nenhum filho, simplesmente remove o nodo
128:             else if (nodo.leftChild == null && nodo.rightChild == null) {
129:                     if (nodo.parent.leftChild == nodo) {
130:                             nodo.parent.leftChild = null;
131:                             nodo.parent = null;
132:                     } else {
133:                             nodo.parent.rightChild = null;
134:                             nodo.parent = null;
135:                     }
136:             //Se tiver so o filho direito, o avo o adota
137:             } else if ( nodo.leftChild == null && nodo.rightChild != null) {
138:                     if (nodo.parent.leftChild == nodo) {
139:                             nodo.parent.leftChild = nodo.rightChild;
140:                             nodo.rightChild.parent = nodo.parent;
141:                             nodo.parent = null;
142:                     } else {
143:                             nodo.parent.rightChild = nodo.rightChild;
144:                             nodo.rightChild.parent = nodo.parent;
145:                             nodo.parent = null;
146:                     }
147:             }
148:             //Se tiver so o filho esquerdo, o avo o adota
149:             else {
150:                     if (nodo.parent.leftChild == nodo) {
```

```
151:                            nodo.parent.leftChild = nodo.leftChild;
152:                            nodo.leftChild.parent = nodo.parent;
153:                            nodo.parent = null;
154:                    } else {
155:                            nodo.parent.rightChild = nodo.leftChild;
156:                            nodo.leftChild.parent = nodo.parent;
157:                            nodo.parent = null;
158:                    }
159:            }
160:    }
161:    /** Remove o nodo quando e preciso trocar de lugar com o antecessor */
162:    void removeAntecessor() {
163:            //Busca antecessor
164:            HumanTree antecessor = this.leftChild;
165:            while ( antecessor.rightChild != null ) antecessor = antecessor.rightChild;
166:            //Faz a troca
167:            this.transferInfoFrom(antecessor);
168:            this.remove(antecessor);
169:    }
170:    /** Caminha a arvore em pre ordem para retornar os nodos em ordem alfabetica */
171:    void caminhaPreOrdem() {
172:            if ( this.leftChild != null ) this.leftChild.caminhaPreOrdem();
173:            this.imprime();
174:            if ( this.rightChild != null ) this.rightChild.caminhaPreOrdem();
175:    }
176: }
```