Classe Nó (Node)

```java
class No {
    public long item;
    public No dir;
    public No esq;
}
```

Classe Árvore Binária (Tree)

```java
import java.io.*;
import java .util.*;

class Tree {
    private No root; // raiz
    private int qntNo = contarNos(root);


    public Tree() { root=null; } // inicializa arvore

    public boolean inserir(long v) {
        No novo = new No(); // cria um novo Nó
        novo.item = v; // atribui o valor recebido ao item de dados do Nó
        novo.dir = null;
        novo.esq = null;

        if (buscarBool(v)){
            return false;
        }

        if (root == null) root = novo;
        else  { // se nao for a raiz
            No atual = root;
            No anterior;
            while(true) {
                anterior = atual;
                if (v <= atual.item) { // ir para esquerda
                    atual = atual.esq;
                    if (atual == null) {
                        anterior.esq = novo;
                        return false;
                    }
                }  // fim da condição ir a esquerda
                else { // ir para direita
                    atual = atual.dir;
                    if (atual == null) {
                        anterior.dir = novo;
                        return false;
                    }
                } // fim da condição ir a direita
            } // fim do laço while
        } // fim do else não raiz
        return false;
    }

    public boolean buscarBool(long chave) {
        if (root == null) return false; // se arvore vazia
        No atual = root;  // começa a procurar desde raiz
        while (atual.item != chave) { // enquanto nao encontrou
            if(chave < atual.item )
                atual = atual.esq; // caminha para esquerda
            else atual = atual.dir; // caminha para direita
            if (atual == null) return false; // encontrou uma folha -> sai
```

```
    } // fim laço while
    return true; // terminou o laço while e chegou aqui é pq encontrou item
}

public No buscar(long chave) {
    if (root == null) return null; // se arvore vazia
    No atual = root;  // começa a procurar desde raiz
    while (atual.item != chave) { // enquanto nao encontrou
        if(chave < atual.item )
            atual = atual.esq; // caminha para esquerda
        else atual = atual.dir; // caminha para direita
        if (atual == null) return null; // encontrou uma folha -> sai
    } // fim laço while
    return atual; // terminou o laço while e chegou aqui é pq encontrou item
}


public boolean remover(long v) {
    if (root == null) return false; // se arvore vazia

    No atual = root;
    No pai = root;
    boolean filho_esq = true;

    // ****** Buscando o valor **********
    while (atual.item != v) { // enquanto nao encontrou
        pai = atual;
        if(v < atual.item ) { // caminha para esquerda
            atual = atual.esq;
            filho_esq = true; // é filho a esquerda? sim
        }
        else { // caminha para direita
            atual = atual.dir;
            filho_esq = false; // é filho a esquerda? NAO
        }
        if (atual == null) return false; // encontrou uma folha -> sai
    } // fim laço while de busca do valor

    // ***************************************************************
    // se chegou aqui quer dizer que encontrou o valor (v)
    // "atual": contem a referencia ao No a ser eliminado
    // "pai": contem a referencia para o pai do No a ser eliminado
    // "filho_esq": é verdadeiro se atual é filho a esquerda do pai
    // ***************************************************************

    // Se nao possui nenhum filho (é uma folha), elimine-o
    if (atual.esq == null && atual.dir == null) {
        if (atual == root ) root = null; // se raiz
        else if (filho_esq) pai.esq = null; // se for filho a esquerda do pai
        else pai.dir = null; // se for filho a direita do pai
    }

    // Se é pai e nao possui um filho a direita, substitui pela subarvore a direita
    else if (atual.dir == null) {
        if (atual == root) root = atual.esq; // se raiz
        else if (filho_esq) pai.esq = atual.esq; // se for filho a esquerda do pai
        else pai.dir = atual.esq; // se for filho a direita do pai
    }

    // Se é pai e nao possui um filho a esquerda, substitui pela subarvore a esquerda
    else if (atual.esq == null) {
        if (atual == root) root = atual.dir; // se raiz
        else if (filho_esq) pai.esq = atual.dir; // se for filho a esquerda do pai
        else pai.dir = atual.dir; // se for  filho a direita do pai
    }
```

```
        // Se possui mais de um filho, se for um avô ou outro grau maior de parentesco
        else {
            No sucessor = no_sucessor(atual);
            // Usando sucessor que seria o Nó mais a esquerda da subarvore a direita do No que
deseja-se remover
            if (atual == root) root = sucessor; // se raiz
            else if(filho_esq) pai.esq = sucessor; // se for filho a esquerda do pai
            else pai.dir = sucessor; // se for filho a direita do pai
            sucessor.esq = atual.esq; // acertando o ponteiro a esquerda do sucessor agora que
ele assumiu
            // a posição correta na arvore
        }

        return true;
    }

    // O sucessor é o Nó mais a esquerda da subarvore a direita do No que foi passado como
parametro do metodo
    public No no_sucessor(No apaga) { // O parametro é a referencia para o No que deseja-se
apagar
        No paidosucessor = apaga;
        No sucessor = apaga;
        No atual = apaga.dir; // vai para a subarvore a direita

        while (atual != null) { // enquanto nao chegar no Nó mais a esquerda
            paidosucessor = sucessor;
            sucessor = atual;
            atual = atual.esq; // caminha para a esquerda
        }

        if (sucessor != apaga.dir) { // se sucessor nao é o filho a direita do Nó que deverá
ser eliminado
            paidosucessor.esq = sucessor.dir; // pai herda os filhos do sucessor que sempre
serão a direita
            // lembrando que o sucessor nunca poderá ter filhos a esquerda, pois, ele sempre
será o
            // Nó mais a esquerda da subarvore a direita do Nó apaga.
            // lembrando também que sucessor sempre será o filho a esquerda do pai

            sucessor.dir = apaga.dir; // guardando a referencia a direita do sucessor para
            // quando ele assumir a posição correta na arvore
        }
        return sucessor;
    }

    public void caminhar() {
        System.out.print("\n Exibindo em ordem: ");
        inOrder(root);
        System.out.print("\n Exibindo em pos-ordem: ");
        posOrder(root);
        System.out.print("\n Exibindo em pre-ordem: ");
        preOrder(root);
        System.out.print("\n Altura da arvore: " + altura(root));
        System.out.print("\n Quantidade de folhas: " + folhas(root));
        System.out.print("\n Quantidade de Nós: " + contarNos(root));
        if (root != null ) {  // se arvore nao esta vazia
            System.out.print("\n Valor minimo: " + min().item);
            System.out.println("\n Valor maximo: " + max().item);
        }
    }

    public void inOrder(No atual) {
        if (atual != null) {
            inOrder(atual.esq);
```

```
            System.out.print(atual.item + " ");
            inOrder(atual.dir);
        }
    }

    public void preOrder(No atual) {
        if (atual != null) {
            System.out.print(atual.item + " ");
            preOrder(atual.esq);
            preOrder(atual.dir);
        }
    }

    public void posOrder(No atual) {
        if (atual != null) {
            posOrder(atual.esq);
            posOrder(atual.dir);
            System.out.print(atual.item + " ");
        }
    }

    public int altura(No atual) {
        if(atual == null || (atual.esq == null && atual.dir == null))
            return 0;
        else {
            if (altura(atual.esq) > altura(atual.dir))
                return ( 1 + altura(atual.esq) );
            else
                return ( 1 + altura(atual.dir) );
        }
    }

    public int folhas(No atual) {
        if(atual == null) return 0;
        if(atual.esq == null && atual.dir == null) return 1;
        return folhas(atual.esq) + folhas(atual.dir);
    }

    public int contarNos(No atual) {
        if(atual == null)  return 0;
        else return ( 1 + contarNos(atual.esq) + contarNos(atual.dir));
    }

    public No min() {
        No atual = root;
        No anterior = null;
        while (atual != null) {
            anterior = atual;
            atual = atual.esq;
        }
        return anterior;
    }

    public No max() {
        No atual = root;
        No anterior = null;
        while (atual != null) {
            anterior = atual;
            atual = atual.dir;
        }
        return anterior;
    }

}
```

Classe Aplicação (Main)

```
import java.util.Scanner;
import java.util.Random;


class Main {
    public static void main(String[] args) {
        Scanner le = new Scanner(System.in);

        //Arvore para uso do Menu
        Tree arvMenu = new Tree();

        Tree arv1 = new Tree();
        Tree arv2 = new Tree();
        Tree arv3 = new Tree();
        Tree arv4 = new Tree();
        Tree arv5 = new Tree();
        Tree arv6 = new Tree();
        Tree arv7 = new Tree();
        Tree arv8 = new Tree();

        Random gerador = new Random();

        //Inserção de Valores Aleatórios

        for (int i = 0; i <= 1000; i++){
            arv1.inserir(gerador.nextInt(1000));
        }

        for (int i = 0; i <= 10000; i++){
            arv2.inserir(gerador.nextInt(1000));
        }

        for (int i = 0; i <= 100000; i++){
            arv3.inserir(gerador.nextInt(1000));
        }

        for (int i = 0; i <= 1000000; i++){
            arv4.inserir(gerador.nextInt(1000));
        }

        // Inserção de valores crescentes

        for (int i = 0; i <= 1000; i++){
            arv5.inserir(i);
        }

        for (int i = 0; i <= 10000; i++){
            arv6.inserir(i);
        }

        for (int i = 0; i <= 100000; i++){
            arv7.inserir(i);
        }

        for (int i = 0; i <= 1000000; i++){
            arv8.inserir(i);
        }

        int opcao;
        long x;
        System.out.print("\n Programa Arvore binaria de long");
        do {
```

```
            System.out.print("\n**********************************");
            System.out.print("\nEntre com a opcao:");
            System.out.print("\n ----1: Inserir");
            System.out.print("\n ----2: Excluir");
            System.out.print("\n ----3: Pesquisar");
            System.out.print("\n ----4: Exibir");
            System.out.print("\n ----5: Sair do programa");
            System.out.print("\n**********************************");
            System.out.print("\n-> ");
            opcao = le.nextInt();
            switch(opcao) {
                case 1: {
                    System.out.print("\n Informe o valor (long) -> ");
                    x = le.nextLong();
                    arvMenu.inserir(x);
                    break;
                }
                case 2: {
                    System.out.print("\n Informe o valor (long) -> ");
                    x = le.nextLong();
                    if ( !arvMenu.remover(x) )
                        System.out.print("\n Valor nao encontrado!");
                    break;
                }
                case 3: {
                    System.out.print("\n Informe o valor (long) -> ");
                    x = le.nextLong();
                    if( arvMenu.buscar(x) != null )
                        System.out.print("\n Valor Encontrado");
                    else
                        System.out.print("\n Valor nao encontrado!");
                    break;
                }
                case 4: {
                    arvMenu.caminhar();
                    break;
                }
            } // fim switch
        } while(opcao != 5);
    }
}
```