LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA I

BINARY SEARCH TREE

DISUSUN OLEH :

KELVIN HERWANDA TANDRIO

M0514027

ASISTEN :

Alfath Prabanuadhi	**M05130**
Della Fitrayani Budiono	**M05130**
Ig.Donny Fernando	**M05130**
Irene Patasik	**M05130**

JURUSAN INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAUHAN ALAM

UNIVERSITAS SEBELAS MARET

SURAKARTA

Rabu, 22 April 2015

BAB I
SCREEN SHOOT PROGRAM

Bagian ini akan membuat program *tree* yang berbasis *Binary Search Tree*. Jenis *tree* ini yang mengisyaratkan bahwa tiap akar harus punya maksimal 2 elemen atau *leaf* dan antara cabang kiri dengan nilai subkiri pada *tree* harus lebih kecil dari akar dan cabang kanan dengan nilai subkanan pada *tree* harus lebih besar dari akar. Berikut ini adalah bentuk program *Binary Search Tree*.

Program 1 : Kelas *AplikasiBST*

```
1   package Tugas_Praktikum6;
2
3   import java.util.Scanner;
4   public class AplikasiBST {
5       public static void main (String [] args) {
6           int a;
7           Scanner masukan = new Scanner (System.in);
8           ProsesBST BST = new ProsesBST ();
9           MenuPilihan();
10          do {
11              System.out.print ("Silahkan Pilih :");
12              int pilih = masukan.nextInt();
13              switch (pilih) {
14                  case 1  : System.out.print ("Masukan Angka : ");
15                            a = masukan.nextInt();
16                            BST.insert(a);
17                            break;
18                  case 2  : System.out.print ("Masukan Angka Yang "+
19                              "Ingin Dihapus : ");
20                            a = masukan.nextInt();
21                            BST.delete(a);
22                            break;
23                  case 3  : System.out.print ("Masukan Angka : ");
24                            a = masukan.nextInt();
25                            BST.find(a);
26                            break;
27                  case 4  : BST.deleteAll();
28                            break;
29                  case 5  : System.out.println ("Hasil Tampilan Pre-order :");
30                            BST.preorder();
31                            break;
32                  case 6  : System.out.println ("Hasil Tampilan In-order :");
33                            BST.inorder();
34                            break;
35                  case 7  : System.out.println ("Hasil Tampilan Post-order : ");
36                            BST.postorder();
37                            break;
38                  case 8  : System.out.println ("\nBanyak Daun = "+BST.leaf());
39                            break;
40                  case 9  : System.out.println ("\nBanyak node = "+BST.getUkuran());
41                            break;
42                  case 10 : System.out.print ("Masukan Angka : ");
43                            a = masukan.nextInt();
44                            BST.CariTerdekat(a);
45                            break;
```

```
                case 11 : System.out.println ("See You Next Time\n");
                          System.exit(0);
                          break;
                default : System.out.println ("Maaf, pilihan anda salah\n");
            }
        }
        while (true);
    }
    public static void MenuPilihan () {
        System.out.println ("Menu :");
        System.out.println ("1  = Masukan Data");
        System.out.println ("2  = Hapus Data");
        System.out.println ("3  = Mencari Data Yang Diinput");
        System.out.println ("4  = Menghapus Semua Data");
        System.out.println ("5  = Menampilkan Pohon secara Pre-order");
        System.out.println ("6  = Menampilkan Pohon secara In-order");
        System.out.println ("7  = Menampilkan Pohon secara Post-order");
        System.out.println ("8  = Jumlah daun pada pohon");
        System.out.println ("9  = Jumlah node pada pohon");
        System.out.println ("10 = Mencari data terdekat dari inputan");
        System.out.println ("11 = Keluar");
    }
}
```

Program 2 : Kelas *ProsesBST*

```
package Tugas_Praktikum6;

public class ProsesBST {
    private Nodeclass akar;
    private static int ukuran = 0;
    public static int daun = 0;

    public boolean kosong() {
        return (akar == null);
    }
    public int getUkuran () {
        return ukuran;
    }
    public int leaf () {
        prosesLeaf (akar);
        return daun;
    }
    public void prosesLeaf (Nodeclass cabang) {
        if (cabang == akar)
            daun = 0;
        if (cabang != null) {
            if (cabang.kiri == null && cabang.kanan == null)
                daun++;
            prosesLeaf (cabang.kiri);
            prosesLeaf (cabang.kanan); }
    }
    public void insert (int data) {
        Nodeclass input = new Nodeclass (data);
        if (kosong()) {
            akar = input;
```

```
            this.ukuran++; }
        else {
            Nodeclass ortu = null;
            Nodeclass sekarang = akar;
            while (sekarang != null) {
                ortu = sekarang;
                if (sekarang.angka < data)
                    sekarang = sekarang.kanan;
                else
                    sekarang = sekarang.kiri; }
            if (ortu.angka < data)
                ortu.kanan = input;
            else
                ortu.kiri = input;
            input.orangtua = ortu;
            this.ukuran++;  }
        System.out.println ("Input data sukses\n");
    }
    public boolean find (int data) {
        boolean found = false;
        Nodeclass ortu = null;
        Nodeclass sekarang = akar;
        while (sekarang != null) {
            ortu = sekarang;
            if (sekarang.angka == data) {
                found = true;
                System.out.println ("Elemen "+sekarang.angka+" ada\n");
                break; }
            else if (sekarang.angka < data)
                sekarang = sekarang.kanan;
            else if (sekarang.angka > data)
                sekarang = sekarang.kiri;
            else
                System.out.println ("Data not found\n"); }
        return found;
    }
    public void inorder () {
        inorder (akar);
    }
    protected void inorder (Nodeclass akar) {
        if (akar == null)
            return;
        inorder (akar.kiri);
        System.out.print(akar.angka+" ");
        inorder (akar.kanan);
    }
    public void postorder () {
        postorder (akar);
    }
    protected void postorder (Nodeclass akar) {
        if (akar == null)
            return;
        postorder (akar.kiri);
        postorder (akar.kanan);
        System.out.print(akar.angka+" ");
    }
    public void preorder () {
        preorder (akar);
    }
    protected void preorder (Nodeclass akar) {
```

```
91          if (akar == null)
92              return;
93          System.out.print(akar.angka+" ");
94          preorder (akar.kiri);
95          preorder (akar.kanan);
96      }
97      public void delete (int data) {
98          Nodeclass ortu = null;
99          Nodeclass sekarang = akar;
100         if (kosong())
101             System.out.println ("Data tree kosong\n");
102         else {
103             boolean found = find (data);
104             if (found)
105                 akar = prosesDelete (akar, data);
106             else
107                 System.out.println ("Data tidak ditemukan"); }
108     }
109     public Nodeclass prosesDelete (Nodeclass akar, int data) {
110         Nodeclass x, y, z;
111         if(akar.getData() == data) {
112             Nodeclass right, left;
113             left = akar.getKiri();
114             right = akar.getKanan();
115             if (left == null && right == null)
116                 return null;
117             else if (left == null) {
118                 x = right;
119                 return x; }
120             else if (right == null) {
121                 x = left;
122                 return x; }
123             else {
124                 y = TemukanMin(right);
125                 x = right;
126                 while (x.getKiri() != null)
127                     x = x.getKiri();
128                 x.setKiri(left);
129                 return y; } }
130         if (data < akar.getData()) {
131             z = prosesDelete(akar.getKiri(), data);
132             akar.setKiri(z);  }
133         else {
134             z = prosesDelete(akar.getKanan(), data);
135             akar.setKanan(z); }
136         System.out.println ("Data "+data+" telah dihapus");
            this.ukuran--;
138         return akar;
139     }
140     public Nodeclass TemukanMin (Nodeclass t) {
141         if (t == null)
142             return t;
143         while (t.kiri != null)
144             t = t.kiri;
145         return t;
146     }
147     public void deleteAll () {
148         ukuran = 0;
149         akar = null;
150         System.out.println ("Semua Data Telah Dihapus");
```

```
151     }
152     public void CariTerdekat (int data) {
153         int prosesCari;
154         if (kosong ())
155             System.out.println ("Data tree kosong");
156         else {
157             prosesCari = Cari (data, akar, 0);
158             if (prosesCari != 0)
159                 System.out.println ("\nData yang terdekat adalah "+prosesCari);
160         }
161     }
162     public int Cari (int c, Nodeclass t, int a) {
163         int ct = 0;
164         boolean found = false;
165         while (t != null) {
166             if ((c+a)==t.angka || (c-a)==t.angka) {
167                 ct = t.angka;
168                 found = true;
169                 break;  }
170             else {
171                 if (c < t.angka)
172                     t = t.kiri;
173                 else if (c > t.angka)
174                     t = t.kanan; }  }
175         if (!found) {
176             a++;
177             ct = Cari (c, akar, a);  }
178         return ct;
179     }
180 }
```

Program 3 : Kelas *Nodeclass*

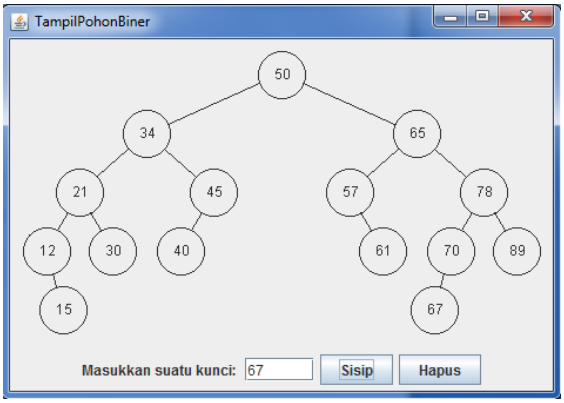
```
1   package Tugas_Praktikum6;
2
3   public class Nodeclass {
4       int angka;
5       Nodeclass orangtua, kiri, kanan;
6
7       public Nodeclass (int data) {
8           this.angka = data;
9           orangtua = kiri = kanan = null;
10      }
11      public Nodeclass getKiri () {
12          return kiri;
13      }
14      public Nodeclass getKanan () {
15          return kanan;
16      }
17      public int getData () {
18          return angka;
19      }
20      public void setKiri (Nodeclass n) {
21          kiri = n;
22      }
23      public void setKanan (Nodeclass n) {
24          kanan = n;
25      }
26  }
```

Hasil Running Program :
Inputan : 50, 34, 65, 21, 45, 57, 78, 12, 30, 40, 61, 70, 89, 15, 67.
Delete : 65, 45

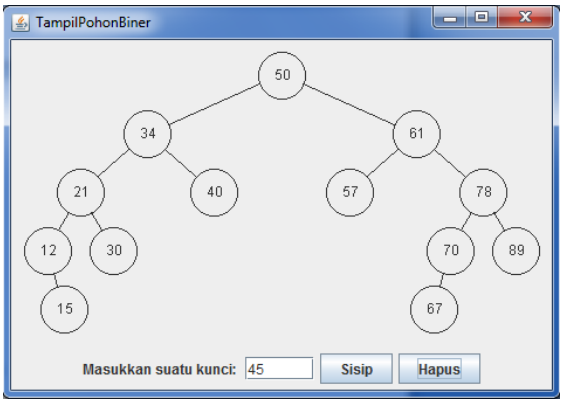
```
run:
Menu :
1  = Masukan Data
2  = Hapus Data
3  = Mencari Data Yang Diinput
4  = Menghapus Semua Data
5  = Menampilkan Pohon secara Pre-order
6  = Menampilkan Pohon secara In-order
7  = Menampilkan Pohon secara Post-order
8  = Jumlah daun pada pohon
9  = Jumlah node pada pohon
10 = Mencari data terdekat dari inputan
11 = Keluar
Silahkan Pilih :1
Masukan Angka : 50
Input data sukses

Silahkan Pilih :1
Masukan Angka : 34
Input data sukses

Silahkan Pilih :1
Masukan Angka : 65
Input data sukses

Silahkan Pilih :1
Masukan Angka : 21
Input data sukses

Silahkan Pilih :1
Masukan Angka : 45
Input data sukses

Silahkan Pilih :1
Masukan Angka : 57
Input data sukses

Silahkan Pilih :1
Masukan Angka : 78
Input data sukses

Silahkan Pilih :1
Masukan Angka : 12
Input data sukses

Silahkan Pilih :1
Masukan Angka : 30
Input data sukses
```

```
Silahkan Pilih :1
Masukan Angka : 40
Input data sukses

Silahkan Pilih :1
Masukan Angka : 61
Input data sukses

Silahkan Pilih :1
Masukan Angka : 70
Input data sukses

Silahkan Pilih :1
Masukan Angka : 89
Input data sukses

Silahkan Pilih :1
Masukan Angka : 15
Input data sukses

Silahkan Pilih :1
Masukan Angka : 67
Input data sukses

Silahkan Pilih :3
Masukan Angka : 70
Elemen 70 ada
Silahkan Pilih :3
Masukan Angka : 30
Elemen 30 ada

Silahkan Pilih :3
Masukan Angka : 14
Silahkan Pilih :5
Hasil Tampilan Pre-order :
50 34 21 12 15 30 45 40 65 57 61 78 70 67 89 Silahkan Pilih :6
Hasil Tampilan In-order :
12 15 21 30 34 40 45 50 57 61 65 67 70 78 89 Silahkan Pilih :7
Hasil Tampilan Post-order :
15 12 30 21 40 45 34 61 57 67 70 89 78 65 50 Silahkan Pilih :8

Banyak Daun = 6
Silahkan Pilih :9

Banyak node = 15
Silahkan Pilih :10
Masukan Angka : 56

Data yang terdekat adalah 57
Silahkan Pilih :10
Masukan Angka : 81

Data yang terdekat adalah 78
Silahkan Pilih :2
Masukan Angka Yang Ingin Dihapus : 65
Elemen 65 ada
```

```
Data 65 telah dihapus
Silahkan Pilih :2
Masukan Angka Yang Ingin Dihapus : 45
Elemen 45 ada

Data 45 telah dihapus
Data 45 telah dihapus
Silahkan Pilih :5
Hasil Tampilan Pre-order :
50 34 21 12 15 30 40 67 57 61 Silahkan Pilih :6
Hasil Tampilan In-order :
12 15 21 30 34 40 50 57 61 67 Silahkan Pilih :7
Hasil Tampilan Post-order :
15 12 30 21 40 34 61 57 67 50 Silahkan Pilih :8

Banyak Daun = 4
Silahkan Pilih :9

Banyak node = 12
Silahkan Pilih :4
Semua Data Telah Dihapus
Silahkan Pilih :5
Hasil Tampilan Pre-order :
Silahkan Pilih :6
Hasil Tampilan In-order :
Silahkan Pilih :7
Hasil Tampilan Post-order :
Silahkan Pilih :11
See You Next Time
```



Gambar 1 : Hasil Inputan Pada Proses Program di Atas



Gambar 2 : Hasil Delete 65 dan 45 Pada Proses Program di Atas

BAB II
ANALISIS DAN ALGORITMA PROGRAM

Pada pemrograman ini akan dibuat *tree* berbasis *Binary Search Tree*. Hal yang utama dalam pembuatan jenis *tree* ini ada 3 hal yaitu sebagai berikut : proses input, delele, dan yang paling utama adalah konsep pembuatan *tree* yang tiap orangtua harus maksimal punya 2 anak atau *leaf*. Pada program ini akan menggunakan prinsip *nodeclass*. *Nodeclass* ini diberikan 3 macam yaitu *orangtua, kiri,* dan *kanan*. Penggunaan node *orangtua* digunakan untuk elemen yang inputa pertama menjadi akar yang kemudian menjadi elemen *orangtua* saat sudah punya 2 anak dan tiap anak punya cabang lagi, node *kanan* digunakan untuk menyimpan elemen yang diinputkan pada cabang *kanan*, dan node *kiri* untuk menyimpan elemen yang diinputkan pada cabang *kiri*. Program ini dibuat tiga kelas yaitu kelas *AplikasiBST*, kelas *ProsesBST*, dan kelas *Nodeclass*. Kelas *AplikasiBST* sebagai program main dan untuk memilih program yang akan dijalankan, kelas *ProsesBST* untuk menjalankan program *Binary Search Tree*, dan kelas *Nodeclass* sebagai variabel pembantu seperti proses *linked list*. Berikut ini adalah bentuk program pada kelas *AplikasiBST* :

```
int a;
Scanner masukan = new Scanner (System.in);
ProsesBST BST = new ProsesBST ();
MenuPilihan();
```

1. Program diawali dengan menginisialisasi variabel *a* dalam bentuk integer kemudian dibuat variabel *masukan* sebagai fungsi Scanner pada *import java.util.Scanner* dan membuat objek *BST* untuk kelas *ProsesBST*.

```
public static void MenuPilihan () {
    System.out.println ("Menu :");
    System.out.println ("1  = Masukan Data");
    System.out.println ("2  = Hapus Data");
    System.out.println ("3  = Mencari Data Yang Diinput");
    System.out.println ("4  = Menghapus Semua Data");
    System.out.println ("5  = Menampilkan Pohon secara Pre-order");
    System.out.println ("6  = Menampilkan Pohon secara In-order");
    System.out.println ("7  = Menampilkan Pohon secara Post-order");
    System.out.println ("8  = Jumlah daun pada pohon");
    System.out.println ("9  = Jumlah node pada pohon");
    System.out.println ("10 = Mencari data terdekat dari inputan");
    System.out.println ("11 = Keluar");
}
```

2. Selanjutnya program masuk ke method *MenuPilihan* yang nantinya akan menampilkan bentuk output seperti di bawah ini :

```
Menu :
1 = Masukan Data
2 = Hapus Data
3 = Mencari Data Yang Diinput
4 = Menghapus Semua Data
5 = Menampilkan Pohon secara Pre-order
```

6 = Menampilkan Pohon secara In-order
7 = Menampilkan Pohon secara Post-order
8 = Jumlah Daun Pada Pohon
9 = Jumlah Node Pada Pohon
10= Mencari data terdekatan dari inputan
11= Keluar

3. Program selanjutnya akan masuk ke proses *do-while* untuk melakukan pemilihan sebuah program yang mau dijalankan dalam *switch* dengan variabel *pilih*.

```
switch (pilih) {
    case 1  : System.out.print ("Masukan Angka : ");
              a = masukan.nextInt();
              BST.insert(a);
              break;
    case 2  : System.out.print ("Masukan Angka Yang "+
                "Ingin Dihapus : ");
              a = masukan.nextInt();
              BST.delete(a);
              break;
    case 3  : System.out.print ("Masukan Angka : ");
              a = masukan.nextInt();
              BST.find(a);
              break;
    case 4  : BST.deleteAll();
              break;
    case 5  : System.out.println ("Hasil Tampilan Pre-order :");
              BST.preorder();
              break;
    case 6  : System.out.println ("Hasil Tampilan In-order :");
              BST.inorder();
              break;
    case 7  : System.out.println ("Hasil Tampilan Post-order : ");
              BST.postorder();
              break;
    case 8  : System.out.println ("\nBanyak Daun = "+BST.leaf());
              break;
    case 9  : System.out.println ("\nBanyak node = "+BST.getUkuran());
              break;
  case 10 : System.out.print ("Masukan Angka : ");
            a = masukan.nextInt();
            BST.CariTerdekat(a);
            break;
  case 11 : System.out.println ("See You Next Time\n");
            System.exit(0);
            break;
  default : System.out.println ("Maaf, pilihan anda salah\n");
```

4. Dalam *switch* ini disediakan 11 case untuk memilih program mana yang akan dijalankan. Case tersebut yaitu sebagai berikut :

Case 1 : untuk menginput angka sebagai elemen pohon.
Case 2 : untuk menghapus elemen yang diinputkan.
Case 3 : untuk mencari elemen pada pohon yang diinputkan.
Case 4 : untuk menghapus semua elemen yang sudah diinputkan.
Case 5 : untuk menampilkan elemen dengan susunan pre-order.
Case 6 : untuk menampilkan elemen dengan susunan in-order.
Case 7 : untuk menampilkan elemen dengan susunan post-order.
Case 8 : untuk mengetahui banyak daun pada pohon yang terdiri dari elemen – elemen.
Case 9 : untuk mengetahui banyak node atau elemen pada pohon.
Case 10 : untuk mengetahui letak elemen yang terdekat dengan angka yang diinputkan.
Case 11 : untuk memberhentikan program atau program keluar.

5. Proses *do-while* ini akan berjalan terus selama kondisi *while* tidak memenuhi yaitu saat membawa nilai salah atau *false* dan nilai tersebut diperoleh dari program case 11 yang pada *System.exit(0)* bernilai 0 yang artinya bernilai *false*.

```
public class Nodeclass {
    int angka;
    Nodeclass orangtua, kiri, kanan;

    public Nodeclass (int data) {
        this.angka = data;
        orangtua = kiri = kanan = null;
    }
    public Nodeclass getKiri () {
        return kiri;
    }
    public Nodeclass getKanan () {
        return kanan;
    }
    public int getData () {
        return angka;
    }
    public void setKiri (Nodeclass n) {
        kiri = n;
    }
    public void setKanan (Nodeclass n) {
        kanan = n;
    }
}
```

Dalam proses pemilihan case pada kelas *AplikasiBST*, program akan masuk ke kelas *ProsesBST* untuk menjalankan program *Binary Search Tree* dengan menggunakan objek *BST*. Dalam kelas *ProsesBST* akan menggunakan Node dengan nama variabel *Nodeclass* yang bentuk algoritma seperti *linked list*. Dilihat dari bentuk program di atas, ada beberapa konstraktor dan method dalam penggunaan Node ini. kelas Node ini bernama kelas *Nodeclass*. Dalam kelas ini diberikan variabel *angka* dalam bentuk integer dan variabel *orangtua, kiri,* dan *kanan* dalam bentuk nama kelas sendiri yaitu *Nodeclass*. Kelas ini dibuat tiga konstruktor da tiga method. Tiga kontruktor tersebut adalah :

a. Konstruktor dengan parameter variabel *data* dalam bentuk integer. Dalam kontruktor ini diberikan *data* akan dijadikan variabel *angka* dan penggunaan *this* menunjukkan bahwa nilai *angka* akan disalurkan ke atas pada *int angka*.
b. Konstruktor dengan nama *getKiri* yang akan mengembalikan nilai variabel *kiri*.
c. Konstruktor dengan nama *getKanan* yang akan mengembalikan nilai variabel *kanan*.

Untuk tiga method tersebut dapat dijelaskan sebagai berikut :

a. Method *getData* yang akan mengembalikan nilai *angka* dalam bentuk integer.
b. Method *getKiri* dengan parameter *n* dalam bentuk Nodeclass akan membawa nilai *n* ke variabel *kiri*.
c. Method *getKanan* dengan parameter *n* dalam bentuk Nodeclass akan membawa nilai *n* ke variabel *kanan*.

Dari bentuk di atas akan digunakan dalam proses jalan program pada kelas *ProsesBST* yang akan saling menghubungkan objek dengan kelas *Nodeclass*.

```
private Nodeclass akar;
private static int ukuran = 0;
public static int daun = 0;

public boolean kosong() {
    return (akar == null);
}
```

Untuk proses program *Binary Search Tree* ini akan dilakukan dengan proses penggunaan *switch* untuk melakukan proses ini yang akan masuk ke kelas *ProsesBST* dengan objek *BST*. Pada kelas *ProsesBST* diberikan variabel *akar* dalam bentuk Nodeclass dan *ukuran* dan *daun* yang dibentuk dalam integer dengan keadaan static yang bernilai awalan 0. Berikut ini adalah uraikan dari tiap case yaitu sebagai berikut :

```
public void insert (int data) {
    Nodeclass input = new Nodeclass (data);
    if (kosong()) {
        akar = input;
        this.ukuran++; }
    else {
        Nodeclass ortu = null;
        Nodeclass sekarang = akar;
        while (sekarang != null) {
            ortu = sekarang;
            if (sekarang.angka < data)
                sekarang = sekarang.kanan;
            else
                sekarang = sekarang.kiri; }
        if (ortu.angka < data)
            ortu.kanan = input;
        else
            ortu.kiri = input;
        input.orangtua = ortu;
        this.ukuran++;  }
    System.out.println ("Input data sukses\n");
}
```

1. Untuk case pertama akan menjalankan program inputan angka yang menggunakan variabel *a* dengan objek *masukan* sebagai fungsi untuk masukan nilai akan masuk ke method *insert* dengan parameter *a*. Dilihat bentuk method pada gambar program di bawah ini, maka jalan program dapat diuraikan sebagai berikut :
   - Pada method ini diawali dengan membuat objek *input* pada kelas Nodeclass dengan parameter *data*. kemudian program akan masuk ke kelas tersebut dan masuk ke konstruktor. Dalam konstrukto ini, program akan menyerahkan *data* ke *angka* dan *angka* tersebut akan dinisialisasi pada *int angka* dengan menggunakan *this* yang membawa nilai *angka = data*.
   - Kemudian, program akan dicek dengan *if-else* dengan dikondisikan *data* itu kosong. Jika kosong (statement *if* memenuhi), maka program akan menjalankan penyerahan *input* ke *akar* lalu pada variabel *ukuran* ditambah 1 dan dinisialisasi pada *private static int ukuran = 0*.
   - Program *if-else* ini, jika tidak kosong (statement *else* memenuhi), maka program akan menjalankan dengan awalan *ortu = null* dan *sekarang = akar* dalam bentuk Nodeclass. Selanjutnya dilooping *while* dengan kondisi *sekarang* tidak *null* atau kosong, maka akan masuk dan mengalihkan *sekarang* ke variabel *ortu* dan diproses dengan *if-else*. Jika nilai *sekarang.angka* (artinya nilai sebelumnya pada *akar*) lebih kecil dari *data* (data baru), maka program akan *sekarang = sekarang.kanan*. Pada *sekarang.kanan* ini dimaksud adalah inputan data baru akan dinisialisasi ke dalam bentuk *kanan* pada kelas *Nodeclass* sehingga inputan baru dijadikan variabel *sekarang kanan*. Selanjutnya, jika *if* tidak memenuhi, maka program masuk *else* dan menjalankan *sekarang = sekarang.kiri* yang artinya inputan data baru menjadi variabel *kiri* pada kelas *Nodeclass*. Setelah itu looping lagi sampai kondisi tidak memenuhi.
   - Setelah kondisi tidak memenuhi, maka program akan cek lagi dengan *if-else*. Jika *if* tidak memenuhi dengan kata lain *ortu.angka < data*, maka program akan menjadikan nilai dari *ortu.angka* menjadi *ortu.kanan* yang akan diproses pada kelas *Nodeclass* dengan objek *input* (masuk konstruktor). Begitupula dengan *else* (saat *if* tidak memenuhi), program akan menjadikan *ortu.angka* jadi *ortu.kiri* dan dijadikan variabel *angka* pada kontruktor kelas *Nodeclass* dengan objek *input*.
   - Kemudian nilai *orangtua* pada kelas *Nodeclass* akan bernilai variabel *ortu* dan kemudian nilai *ukuran* bertambah 1. Lalu programm keluar dari *else* dan menampilkan “Input data sukses”.

2. Untuk case kedua akan memproses penghapusan data dari data yang diinputkan dengan variabel *a* yang akan masuk ke method *delete*. Method ini dapat diuraikan sebagai berikut :

```
public void delete (int data) {
    Nodeclass ortu = null;
    Nodeclass sekarang = akar;
    if (kosong())
        System.out.println ("Data tree kosong\n");
    else {
        boolean found = find (data);
        if (found)
            akar = prosesDelete (akar, data);
        else
            System.out.println ("Data tidak ditemukan"); }
}
```

- Program diawali dengan deklarasi *ortu* = *null* dan *sekarang* = *akar* dalam bentuk Nodeclass.
- Diseleksi program apakah kosong. Jika kosong akan menampilkan "Data tree kosong" sedangkan jika tidak, maka akan menjalankan program untuk proses menemukan data pada method *find* yang dideklarasikan dalam bentuk *found*. Jika *found*, maka akan masuk ke method *prosesDelete* dengan parameter *akar* dan *data*. sedagkan, jika tidak, maka akan menampilkan "Data tidak ditemukan" pada statement *else*.
- Pada method *prosesDelete* ini, bentuk program dapat diuraikan sebagai berikut :

```
public Nodeclass prosesDelete (Nodeclass akar, int data) {
    Nodeclass x, y, z;
    if(akar.getData() == data) {
        Nodeclass right, left;
        left = akar.getKiri();
        right = akar.getKanan();
        if (left == null && right == null)
            return null;
        else if (left == null) {
            x = right;
            return x; }
        else if (right == null) {
            x = left;
            return x; }
        else {
            y = TemukanMin(right);
            x = right;
            while (x.getKiri() != null)
                x = x.getKiri();
            x.setKiri(left);
            return y; } }
    if (data < akar.getData()) {
        z = prosesDelete(akar.getKiri(), data);
        akar.setKiri(z);  }
    else {
        z = prosesDelete(akar.getKanan(), data);
        akar.setKanan(z); }
    System.out.println ("Data "+data+" telah dihapus");
    this.ukuran--;
    return akar;
```

a. Program diawali dengan inisialisasi *x*, *y*, *z* dalam bentuk Nodeclass dan dicek apakah *akar.getData( ) == data*. jika memenuhi, maka akan menjalankan pada statement ini sedangkan jika tidak memenuhi, maka akan langsung ke *if* yang lain.
b. Dalam *if* tersebut, program akan mendeklarasikan *right*, *left* dalam bentuk Nodeclass denga bentuk program sesuai di atas.
c. Dibuat 4 kondisi yaitu sebagai berikut :
   - Kondisi *if* : akan mengembalikan nilai *null*.
   - Kondisi *else if* [left = null] : menyalurkan nilai *left* ke *x* dan mengembalikannya.

- Kondisi *else if* [right = null] : menyalurkan nilai *right* ke *x* dan mengembalikannya.
- Kondisi *else* : variabel *y* akan masuk ke method *TemukanMin* dengan parameter *right* dan bentuk method dapat dilihat sebagai berikut :

```
public Nodeclass TemukanMin (Nodeclass t) {
    if (t == null)
        return t;
    while (t.kiri != null)
        t = t.kiri;
    return t;
}
```

  - Method ini akan membawa variabel berupa *t*. Program dicek pada *if* dengan kondisi *t = null*. Jika memenuhi maka akan mengembalikan nilai *t*.
  - Kemudian looping dengan kondisi *t.kiri != null* akan menyerahkan nilai *t.kiri* menjadi *t* sampai kondisi tidak memenuhi.
  - Program akan mengembalikan nilai *t*.
- Program variabel *x = right* dan looping *while* dengan kondisi *x.getKiri ( ) != null* dan program akan menjalankan *x = x.setKiri ( )* dan akan mengembalikan nilai *y*.
- Kemudia dicek dengan *if-else*. Statement ini akan menjalankan lagi pada method ini yaitu proses rekursif pada nilai *z* sesuai bentuk program di atas.
- Setelah proses ini selesai, maka akan menampikan output sesuai pada *System.out.println* dan mengurangi 1 pada *ukuran* dan mengembalikan nilai akar.

3. Untuk case ketiga ini digunakan untuk mencari data yang sudah diinputkan dengan variabel *a* akan masuk ke method *find*. Bentuk method tersebut dapat diuraikan sebagai berikut :

```
public boolean find (int data) {
    boolean found = false;
    Nodeclass ortu = null;
    Nodeclass sekarang = akar;
    while (sekarang != null) {
        ortu = sekarang;
        if (sekarang.angka == data) {
            found = true;
            System.out.println ("Elemen "+sekarang.angka+" ada\n");
            break; }
        else if (sekarang.angka < data)
            sekarang = sekarang.kanan;
        else if (sekarang.angka > data)
            sekarang = sekarang.kiri;
        else
            System.out.println ("Data not found\n"); }
    return found;
}
```

- Program diawali dengan variabel *found* dibuat bernilai salah dan mendeklarasikan *ortu = null* dan *sekarang = akar* dalam bentuk Nodeclass.
- Kemudian, diproses looping dengan kondisi *sekarang* tidak kosong.
- Dalam looping tersebut pertama dibuat *ortu = sekarang* lalu dicek dengan *if-else*.
- Jika *if* memenuhi, maka program akan mengubah nilai *found* menjadi benar dan menampilkan "Elemen (hasil print out *sekarang,angka*) ada" dan di-*break* dengan tujuan program keluar dari proses looping.
- Apabila *if* tidak memenuhi, maka program akan menjalankan statement lain. Bentuk statement lain ini bentuk programnya hampir sama dengan method *insert* sehingga method ini hanya menjalankan program untuk mencari data yang diinput sesuai data yang telah dibuat pada *tree* dengan konsep seperti method *insert*.
- Setelah melakukan proses looping, program akan mengembalikan nilai *found*.

```
public void deleteAll () {
    ukuran = 0;
    akar = null;
    System.out.println ("Semua Data Telah Dihapus");
}
```

4. Untuk case keempat ini akan menghapus semua data yang telah diinputkan pada method *deleteAll*. Bentuk method ini dibuat program *ukuran = 0* dan *akar = null* yang menunjukkan bahwa nilai *ukuran* dan *akar* ini dibuat kosong sehingga saat melakukan jalan program pada method *find, preorder, inorder, postorder* akan tidak ditemukan karena nilainya kosong. Selanjutnya program akan menampilkan "Semua Data Telah Dihapus".

5. Untuk case kelima digunakan untuk menampilkan data *tree* yang sudah diinputkan dalam bentuk *pre-order*. Proses ini akan masuk ke method *preorder* dengan bentuk program seperti di bawah ini. program ini dapat diuraikan sebagai berikut :

```
public void preorder () {
    preorder (akar);
}
protected void preorder (Nodeclass akar) {
    if (akar == null)
        return;
    System.out.print(akar.angka+" ");
    preorder (akar.kiri);
    preorder (akar.kanan);
}
```

- Program ini akan menjalankan method lagi yaitu *preorder* dalam bentuk *protected void* dengan parameter *akar*.
- Proses ini akan dilakukan secara rekursif dengan pemanggilan method sebanyak dua kali yaitu dengan parameter *akar.kiri* dan *akar.kanan* sampai dalam keadaan *akar =null* yang akan menjalankan statement *if* untuk mengembalikan ke asal mula. Rekursif ini berjalan setelah menampilkan nilai output.

6. Untuk case keenam digunakan untuk menampilkan data *tree* yang sudah diinputkan dalam bentuk *in-order*. Bentuk method tersebut dapat diuraikan sebagai berikut :

```
public void inorder () {
    inorder (akar);
}
protected void inorder (Nodeclass akar) {
    if (akar == null)
        return;
    inorder (akar.kiri);
    System.out.print(akar.angka+" ");
    inorder (akar.kanan);
}
```

- Program ini sama bentuknya pada *preorder* yaitu membuat method lagi yaitu *inorder* dalam bentuk *protected void* dengan parameter *akar*.
- Proses ini akan dilakukan secara rekursif dengan pemanggilan method sebanyak dua kali yaitu dengan parameter *akar.kiri* dan *akar.kanan* sampai dalam keadaan *akar =null* yang akan menjalankan statement *if* untuk mengembalikan ke asal mula. Rekursif ini akan dijalankan pertama

7. Untuk case ketujuh digunakan untuk menampilkan data *tree* yang sudah diinputkan dalam bentuk *post-order*. Bentuk method tersebut dapat diuraikan sebagai berikut :

```
public void postorder () {
    postorder (akar);
}
protected void postorder (Nodeclass akar) {
    if (akar == null)
        return;
    postorder (akar.kiri);
    postorder (akar.kanan);
    System.out.print(akar.angka+" ");
}
```

- Program ini akan menjalankan method lagi yaitu *postorder* dalam bentuk *protected void* dengan parameter *akar* sama seperti pada method *preorder* dan *inorder*.
- Proses ini akan dilakukan secara rekursif dengan pemanggilan method sebanyak dua kali yaitu dengan parameter *akar.kiri* dan *akar.kanan* sampai dalam keadaan *akar =null* yang akan menjalankan statement *if* untuk mengembalikan ke asal mula. Rekursif ini akan dilakukan terlebih dahulu sebanyak dua kali baru menampilkan hasil outputnya.

```
public int leaf () {
    prosesLeaf (akar);
    return daun;
}
public void prosesLeaf (Nodeclass cabang) {
    if (cabang == akar)
        daun = 0;
    if (cabang != null) {
        if (cabang.kiri == null && cabang.kanan == null)
            daun++;
        prosesLeaf (cabang.kiri);
        prosesLeaf (cabang.kanan); }
}
```

8. Untuk case kedelapan untuk menentukan jumlah daun pada *tree* yang sudah diinputkan berbagai data. proses ini akan menampilkan “Banyak Daun = ...” yang hasilnya akan masuk ke method *leaf*. Method tersebut dapat diuraikan sebagai berikut :
   - Method ini akan masuk lagi ke method *prosesLeaf* dengan parameter *akar* dalam bentuk Nodeclass.
   - Setelah masuk ke method tersebut, ada dua kondisi *if* yaitu :
     a. Kondisi dimana *cabang == akar* maka program akan menginisialisasikan *daun* = 0.
     b. Kondisi dimana *cabang* tidak kosong program akan melakukan pemanggilan method kembali sebanyak dua kali dengan parameter *cabang.kiri* dan *cabang.kanan* sampai program memnuhi pada kondisi *if* yang akan menambah nilai 1 pada variabel *daun*.

```
public int getUkuran () {
    return ukuran;
}
```

9. Case kesembilan akan menentukan jumlah elemen pada *tree* yang akan masuk ke method *getUkuran* sekaligus menampilkan “ Banyak node = (hasil dari method *getUkuran*)”. Dalam method ini dapat dilihat *return ukuran* yang artinya akan mengembalikan nilai *ukuran* dari hasil proses penginputan dan penghapusan data pada method *insert* dan *delete*.

10. Case kesepeluh untuk mencari data terdekat pada tiap elemen dalam *tree* dari data yang diinputkan dengan variabel *a*. proses ini akan masuk ke method *CariTerdekat* dengan parameter *a*. bentuk methodnya dapat diuraikan sebagai berikut :

```
public void CariTerdekat (int data) {
    int prosesCari;
    if (kosong ())
        System.out.println ("Data tree kosong");
    else {
        prosesCari = Cari (data, akar, 0);
        if (prosesCari != 0)
            System.out.println ("\nData yang terdekat adalah "+prosesCari);
    }
}
```

- Program diawali dengan mendeklarasikan variabel *prosesCari* dalam bentuk integer.
- Selanjutnya, program mengecek dengan statement *if-else*. Jika *if* memenuhi, maka program akan menampilkan "Data tree kosong". Apabila *if* tidak memenuhi, maka akan menjalankan *else* dan nilai *prosesCari* akan masuk ke method *Cari* dengan parameter *data, akar* dan nilai *0*. Setelah itu, program mengecek apakah nilai *prosesCari* tidak sama dengan 0. Jika terpenuhi maka akan menampilkan "Data yang terdekat adalah (hasil output *prosesCari*).
- Dalam mehod *Cari* ini dapat diuraikan sebagai berikut :

```
public int Cari (int c, Nodeclass t, int a) {
    int ct = 0;
    boolean found = false;
    while (t != null) {
        if ((c+a)==t.angka || (c-a)==t.angka) {
            ct = t.angka;
            found = true;
            break;  }
        else {
            if (c < t.angka)
                t = t.kiri;
            else if (c > t.angka)
                t = t.kanan; }  }
    if (!found) {
        a++;
        ct = Cari (c, akar, a);  }
    return ct;
}
```

a. Program diawali dengan inisialisasi variabel *ct* dalam bentuk integer dengan nilai 0 dan dibuat variabel *found* bernilai *false*.
b. Program akan masuk looping *while* dengan kondisi *t* tidak kosong. Dalam looping ini diseleksi pada *if-else*. Jika *if* memenuhi sesuai kondisi di atas, maka program akan menyerahkan nilai *t.angka* ke *ct* kemudian mengembalikan nilai benar pada variabel *found* dan di-*break* untuk keluar looping *while*. Penggunaan *if* ini karena *found = true*, maka program tidak menjalanan statement *if* melainkan langsung mengembalikan nilai *ct*.
c. Jika *if* tidak memenuhi pada looping *while*, maka program akan menjalankan *else* dan dicek lagi pada statement *if-else*. Jika *if* memenuhi, program akan menyerahkan nilai *t.kiri* ke *t* begitu juga pada *else* akan menyerahkan nilai

t.kanan ke *t* apabila *if* tidak memenuhi. Karena menjalankan program *else*, maka akan masuk ke statement *if* setelah looping karena nilai *found* masih salah sehingga nilai *a* bertambah 1 dan memanggil method sendiri untuk menjalankan program methos sampai variabel *found* mengembalikan nilai *true* dalam statement *if* pada looping *while*.

11. Case terakhir akan menjalankan program untuk keluar dari proses looping *do-while* dan program akan berhenti.

Dari proses case ini, dapat dilihat bahwa proses tersebut dapat mengatur masukan data dalam bentuk *Binary Search Tree* yang tentunya diatur dalam proses pemasukan, pencarian, penghapusan, dan lain – lain sehingga bisa memenuhi konsep *tree* BST. Proses case tersebut jika tidak sesuai pada case yang disediakan, maka akan masuk ke *default* dan menampilkan “Maaf, pilihan anda salah”.