ALGORITMA DAN STRUKTUR DATA

“Laporan hasil Praktikum pada Jobsheet 14 “Tree” ”

Oleh:

Hafiz Rizqi Hernanda

NIM (244107020154)

Jurusan Teknologi informasi

Teknik Informatika

Politeknik Negeri Malang

14.2 Kegiatan Praktikum 1

Implementasi Binary Search Tree menggunakan Linked List (100 Menit)

14.2.1 Percobaan 1

Pada percobaan ini akan diimplementasikan Binary Search Tree dengan operasi dasar, dengan menggunakan linked list.

1. Pada Project yang sudah dibuat pada pertemuan sebelumnya, buat package dengan nama Pertemuan14.
2. Tambahkan class-class berikut:

a. Mahasiswa00.java

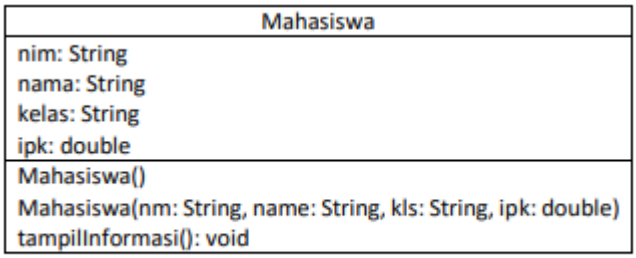
b. Node00.java

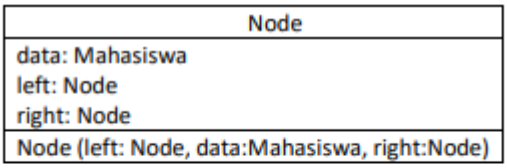
c. BinaryTree00.java

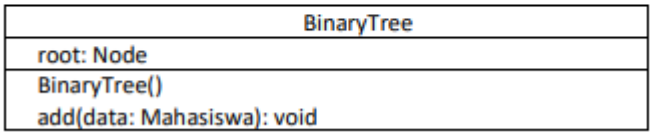
d. BinaryTreeMain00.java

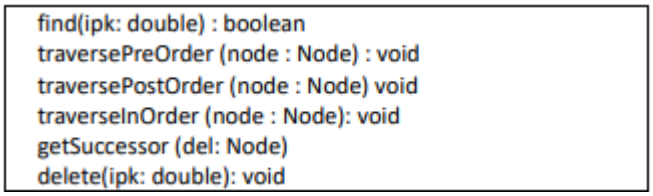
Ganti 00 dengan nomer absen Anda.

3. Implementasikan Class Mahasiswa00, Node00, BinaryTree00 sesuai dengan diagram class berikut ini:









1. Di dalam class Mahasiswa00, deklarasikan atribut sesuai dengan diagram class Mahasiswa di atas. Tambahkan juga konstruktor dan method sesuai diagram di atas.

```java
package Jobsheet14;
public class Mahasiswa11 {
    String nim;
    String nama;
    String kelas;
    double ipk;

    public Mahasiswa11() {}

    public Mahasiswa11(String nim, String nama, String kelas, double ipk) {
        this.nim = nim;
        this.nama = nama;
        this.kelas = kelas;
        this.ipk = ipk;
    }

    public void tampilInformasi() {
        System.out.println("NIM: " + this.nim + " " +
                           "Nama: " + this.nama + " " +
                           "Kelas: " + this.kelas + " " +
                           "IPK: " + this.ipk);
    }
}
```

2. Di dalam class Node00, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.

```java
package Jobsheet14;

public class Node11 {
    Mahasiswa11 mahasiswa;
    Node11 left, right;

    public Node11() {}

    public Node11(Mahasiswa11 mahasiswa) {
        this.mahasiswa = mahasiswa;
        left = right = null;
    }
}
```

3. Di dalam class BinaryTree00, tambahkan atribut root

```java
package Jobsheet14;

public class BinaryTree11 {
    Node11 root;
```

4. Tambahkan konstruktor dan method isEmpty() di dalam class BinaryTree00

```
6       public BinaryTree11() {
7           root = null;
8       }
9
10      public boolean isEmpty() {
11          return root == null;
12      }
```

5. Tambahkan method add() di dalam class BinaryTree00. Node ditambahkan di binary search tree pada posisi sesuai dengan besar nilai IPK mahasiswa. Di bawah ini proses penambahan node tidak dilakukan secara rekursif, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya dengan proses rekursif penulisan kode akan lebih efisien.

```
14      public void add(Mahasiswa11 mahasiswa) {
15          Node11 newNode = new Node11(mahasiswa);
16          if (isEmpty()) {
17              root = newNode;
18          } else {
19              Node11 current = root;
20              Node11 parent = null;
21              while (true) {
22                  parent = current;
23                  if (mahasiswa.ipk < current.mahasiswa.ipk) {
24                      current = current.left;
25                      if (current == null) {
26                          parent.left = newNode;
27                          return;
28                      }
29                  } else {
30                      current = current.right;
31                      if (current == null) {
32                          parent.right = newNode;
33                          return;
34                      }
35                  }
36              }
37          }
38      }
```

6. Tambahkan method find()

```
40      boolean find(double ipk) {
41          boolean result = false;
42          Node11 current = root;
43          while (current != null) {
44              if (current.mahasiswa.ipk == ipk) {
45                  result = true;
46                  break;
47              } else if (ipk > current.mahasiswa.ipk) {
48                  current = current.right;
49              } else {
50                  current = current.left;
51              }
52          }
53          return result;
54      }
```

7. Tambahkan method traversePreOrder(), traverseInOrder() dan traversePostOrder(). Method traverse digunakan untuk mengunjungi dan menampilkan node-node dalam binary tree, baik dalam mode pre-order, in-order maupun post-order.

```
56      void traversalPreOrder(Node11 node) {
57          if (node != null) {
58              node.mahasiswa.tampilInformasi();
59              traversalPreOrder(node.left);
60              traversalPreOrder(node.right);
61          }
62      }
63
64      void traversalInOrder(Node11 node) {
65          if (node != null) {
66              traversalInOrder(node.left);
67              node.mahasiswa.tampilInformasi();
68              traversalInOrder(node.right);
69          }
70      }
71
72      void traversalPostOrder(Node11 node) {
73          if (node != null) {
74              traversalPostOrder(node.left);
75              traversalPostOrder(node.right);
76              node.mahasiswa.tampilInformasi();
77          }
78      }
```

8. Tambahkan method getSuccessor(). Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.

```
80      Node11 getSuccessor(Node11 del) {
81          Node11 successor = del.right;
82          Node11 successorParent = del;
83          while (successor.left != null) {
84              successorParent = successor;
85              successor = successor.left;
86          }
87          if (successor != del.right) {
88              successorParent.left = successor.right;
89              successor.right = del.right;
90          }
91          return successor;
92      }
```

9. Tambahkan method delete(). Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak, cari posisi node yang akan dihapus

```
94      void delete(double ipk) {
95          if(isEmpty()) {
96              System.out.println(x:"Binary Tree Kossong");
97              return;
98          }
99          //cari node (current) yang akan dihapus
100         Node11 parent =  root;
101         Node11 current = root;
102         boolean isLeftChild = false;
103         while (current != null) {
104             if (current.mahasiswa.ipk == ipk) {
105                 break;
106             } else if (ipk < current.mahasiswa.ipk) {
107                 parent = current;
108                 current = current.left;
109                 isLeftChild = true;
110             } else if( ipk > current.mahasiswa.ipk) {
111                 parent = current;
112                 current = current.right;
113                 isLeftChild = false;
114             }
115         }
```

10. Kemudian tambahkan proses penghapusan di dalam method delete() terhadap node current yang telah ditemukan.

```
116         //penghapusan
117         if (current == null) {
118             System.out.println(x:"Data tidak ditemukan");
119             return;
120         }else{
121             //jika tida ada anak (leaf), maka node dihapus
122             if (current.left == null && current.right == null) {
123                 if (current == root) {
124                     root = null;
125                 } else if (isLeftChild) {
126                     parent.left = null;
127                 } else {
128                     parent.right = null;
129                 }
130             } else if (current.left == null) {//jika hanya punya 1 anak (kanan)
131                 if (current == root) {
132                     root = current.right;
133                 } else if (isLeftChild) {
134                     parent.left = current.right;
135                 } else {
136                     parent.right = current.right;
137                 }
```

```
            } else if (current.right == null) {//jika hanya punya 1 anak (kiri)
                if (current == root) {
                    root = current.left;
                } else
                  {
                    if (isLeftChild) {
                        parent.left = current.left;
                    } else {
                        parent.right = current.left;
                    }
                }
            } else {//jika punya 2 anak
                Node11 successor = getSuccessor(current);
                System.out.println(x:"Jika 2 anak, current = ");
                successor.mahasiswa.tampilInformasi();
                if (current == root) {
                    root = successor;
                } else{
                    if (isLeftChild) {
                        parent.left = successor;
                    } else {
                        parent.right = successor;
                    }
                }
                successor.left = current.left;
            }

        }
    }
}
```

11. Buka class BinaryTreeMain00 dan tambahkan method main() kemudian tambahkan kode berikut ini:

```java
package Jobsheet14;

public class BinaryTreeMain11 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTree11 bat = new BinaryTree11();

        bat.add(new Mahasiswa11(nim:"244160121", nama:"Ali", kelas:"A", ipk:3.57));
        bat.add(new Mahasiswa11(nim:"244160121", nama:"Badar", kelas:"B", ipk:3.85));
        bat.add(new Mahasiswa11(nim:"244160185", nama:"Candra", kelas:"C", ipk:3.21));
        bat.add(new Mahasiswa11(nim:"244160220", nama:"Dewi", kelas:"D", ipk:3.54));

        System.out.println(x:"\nDaftar nama mahasiswa(in Order Traversal):");
        bat.traversalInOrder(bat.root);

        System.out.println(x:"\nPencarian data mahasiswa:");
        System.out.print(s:"Cari mahasiswa dengan ipk: 3.54 : ");
        String hasilCari = bat.find(ipk:3.54) ? "Ditemukan" : "Tidak ditemukan";
        System.out.println(hasilCari);

        System.out.print(s:"Cari mahasiswa dengan ipk: 3.22 : ");
        hasilCari = bat.find(ipk:3.22) ? "Ditemukan" : "Tidak ditemukan";
        System.out.println(hasilCari);

        bat.add(new Mahasiswa11(nim:"244160131", nama:"Devi", kelas:"A", ipk:3.72));
        bat.add(new Mahasiswa11(nim:"244160205", nama:"Ehsan", kelas:"D", ipk:3.37));
        bat.add(new Mahasiswa11(nim:"244160170", nama:"Fizi", kelas:"B", ipk:3.46));
        System.out.println(x:"\nDaftar semua mahasiswa setelah penambahan 3 mahasiswa:");
        System.out.println(x:"\nInOrder Traversal:");
        bat.traversalInOrder(bat.root);
        System.out.println(x:"\nPreOrder Traversal:");
        bat.traversalPreOrder(bat.root);
        System.out.println(x:"\nPostOrder Traversal:");
        bat.traversalPostOrder(bat.root);

        System.out.println(x:"\npenghapusan data mahasiswa");
        bat.delete(ipk:3.57);
        System.out.println(x:"\nDaftar mahasiswa setelah penghapusan 1 mahasiswa (in Order):");
        bat.traversalInOrder(bat.root);
    }


}
```

12. Compile dan jalankan class BinaryTreeMain00 untuk mendapatkan simulasi jalannya program binary tree yang telah dibuat.
13. Amati hasil running tersebut

```
Daftar semua mahasiswa (in oder traversal):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85

Pencarian data mahasiswa:
Cari mahasiswa dengan ipk: 3.54 : Ditemukan
Cari mahasiswa dengan ipk: 3.22 : Tidak ditemukan
```

```
Daftar semua mahasiswa setelah penambahan 3 mahasiswa:
InOrder Traversal:
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85

PreOrder Traversal:
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72

PostOrder Traversal:
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
```

```
Penghapusan data mahasiswa

Daftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.85
```

```
Daftar nama mahasiswa(in Order Traversal):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: D IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160121 Nama: Badar Kelas: B IPK: 3.85

Pencarian data mahasiswa:
Cari mahasiswa dengan ipk: 3.54 : Ditemukan
Cari mahasiswa dengan ipk: 3.22 : Tidak ditemukan

Daftar semua mahasiswa setelah penambahan 3 mahasiswa:

InOrder Traversal:
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: D IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160121 Nama: Badar Kelas: B IPK: 3.85

PreOrder Traversal:
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: D IPK: 3.54
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160121 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72

PostOrder Traversal:
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160220 Nama: Dewi Kelas: D IPK: 3.54
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160121 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57

penghapusan data mahasiswa
Jika 2 anak, current =
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
```

```
Daftar mahasiswa setelah penghapusan 1 mahasiswa (in Order):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: D IPK: 3.54
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160121 Nama: Badar Kelas: B IPK: 3.85
PS C:\Regulus\Praktikum-ASD>
```

14.2.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?

Karena **binary search tree (BST)** menyimpan data secara **terurut**. Di setiap node:

- Data di kiri **lebih kecil**, dan
- Data di kanan **lebih besar** dari node itu sendiri.

Jadi, saat mencari data, kita bisa **langsung membandingkan** dan memilih arah (kiri atau kanan), tanpa harus cek semua node satu per satu seperti di binary tree biasa. Ini membuat pencarian **lebih cepat dan efisien**.

2. Untuk apakah di class Node, kegunaan dari atribut left dan right?

Atribut left dan right digunakan untuk menyimpan **referensi** ke **anak kiri** dan **anak kanan** dari sebuah node. Ini yang membentuk struktur pohon:

- left menunjuk ke node yang nilainya **lebih kecil**
- right menunjuk ke node yang nilainya **lebih besar**

3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?

Atribut root adalah **awal** dari seluruh pohon. Dari root, kita bisa mengakses semua node lainnya dengan menelusuri left dan right.

b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?

Saat pertama kali tree dibuat, root masih **belum ada isinya**, jadi nilainya **null** (kosong).

4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

Kalau tree masih kosong (artinya root masih null), maka node pertama yang ditambahkan akan menjadi **root** dari tree tersebut.

5. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

```
parent = current;
if (mahasiswa.ipk < current.mahasiswa.ipk) {
    current = current.left;
    if (current == null) {
        parent.left = newNode;
        return;
    }
} else {
    current = current.right;
    if (current == null) {
        parent.right = newNode;
        return;
    }
}
```

Penjelasan:

- parent = current; Menyimpan node sekarang sebagai induk (parent).
- if (mahasiswa.ipk < current.mahasiswa.ipk) = Cek apakah data baru **lebih kecil** dari node sekarang.
    - Kalau ya, maka pindah ke **anak kiri** (current.left)
    - Jika current.left kosong, maka node baru akan **dipasang di kiri** parent.
- Else, Kalau data baru **lebih besar atau sama**, maka pindah ke **anak kanan**
    - Kalau current.right kosong, node baru akan **dipasang di kanan** parent.

6. Jelaskan langkah-langkah pada method delete() saat menghapus sebuah node yang memiliki dua anak. Bagaimana method getSuccessor() membantu dalam proses ini?

Langkah-langkah delete() untuk node dengan 2 anak:

1. **Cari node yang mau dihapus.**
2. Karena node punya **dua anak**, kita tidak bisa langsung hapus.
3. Kita harus cari **pengganti** yang nilainya mendekati node yang dihapus.
    - Biasanya pakai **successor**, yaitu **nilai terkecil di anak kanan**.
4. Gunakan method getSuccessor() untuk **mendapatkan node tersebut**.
5. Ganti isi node yang mau dihapus dengan isi dari **successor**.
6. Lalu hapus **successor** dari tempat aslinya (karena sekarang sudah dipindahkan ke posisi node yang dihapus).

Peran getSuccessor():

- Menemukan **node pengganti** yang cocok.
- Biasanya dengan **menelusuri anak kanan** lalu ke kiri terus sampai mentok (nilai terkecil di subtree kanan).

14.3 Kegiatan Praktikum 2

Implementasi Binary Tree dengan Array (45 Menit)

14.3.1 Tahapan Percobaan

1. Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method main(), dan selanjutnya akan disimulasikan proses traversal secara inOrder.
2. Buatlah class BinaryTreeArray00 dan BinaryTreeArrayMain00. Ganti 00 dengan nomer absen Anda.
3. Buat atribut data dan idxLast di dalam class BinaryTreeArray00. Buat juga method populateData() dan traverseInOrder()

```java
package Jobsheet14;

public class BinaryTreeArray11 {
    Mahasiswa11[] dataMahasiswa;
    int idxLast;

    public BinaryTreeArray11() {
        this.dataMahasiswa = new Mahasiswa11[10];
    }

    void populateData (Mahasiswa11 dataMhs[], int idxLast) {
        this.dataMahasiswa = dataMhs;
        this.idxLast = idxLast;
    }

    void traverseInOrder(int idxStart) {
        if(idxStart <= idxLast) {
            if (dataMahasiswa[idxStart] != null ) {
                traverseInOrder(2*idxStart+1);
                dataMahasiswa[idxStart].tampilInformasi();
                traverseInOrder(2*idxStart+2);
            }
        }
    }
}
```

4. Kemudian dalam class BinaryTreeArrayMain00 buat method main() dan tambahkan kode seperti gambar berikut ini di dalam method main().

```java
package Jobsheet14;

public class BinaryTreeMain11 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray11 bta = new BinaryTreeArray11();

        Mahasiswa11 mhs1 = new Mahasiswa11(nim:"244160121", nama:"Ali", kelas:"A", ipk:3.57);
        Mahasiswa11 mhs2 = new Mahasiswa11(nim:"244160185", nama:"Candra", kelas:"C", ipk:3.41);
        Mahasiswa11 mhs3 = new Mahasiswa11(nim:"244160221", nama:"Badar", kelas:"B", ipk:3.75);
        Mahasiswa11 mhs4 = new Mahasiswa11(nim:"244160220", nama:"Dewi", kelas:"B", ipk:3.35);
        Mahasiswa11 mhs5 = new Mahasiswa11(nim:"244160131", nama:"Devi", kelas:"A", ipk:3.48);
        Mahasiswa11 mhs6 = new Mahasiswa11(nim:"244160205", nama:"Ehsan", kelas:"D", ipk:3.61);
        Mahasiswa11 mhs7 = new Mahasiswa11(nim:"244160170", nama:"Fizi", kelas:"B", ipk:3.86);

        Mahasiswa11[] dataMahasiswa = {mhs1, mhs2, mhs3, mhs4, mhs5, mhs6, mhs7, null, null, null};
        int idxLast = 6;
        bta.populateData(dataMahasiswa, idxLast);
        System.out.println(x:"\nInOrder Traversal Mahasiswa: ");
        bta.traverseInOrder(idxStart:0);
    }


}
```

5. Jalankan class BinaryTreeArrayMain00 dan amati hasilnya!

```
Inorder Traversal Mahasiswa:
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.35
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.41
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.48
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.61
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.75
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.86
```

```
InOrder Traversal Mahasiswa:
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.35
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.41
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.48
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.61
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.75
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.86
```

14.3.2 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?

- dataMahasiswa adalah **array** yang menyimpan objek-objek Mahasiswa11 sebagai node dari binary tree.
- idxLast adalah **indeks terakhir** dari node yang terisi di dalam array (menunjukkan batas akhir data pada pohon).

2. Apakah kegunaan dari method populateData()?

- **Mengisi array** dataMahasiswa dengan data dari luar (dari parameter dataMhs[]).
- -Menyimpan nilai indeks terakhir (idxLast) untuk membatasi proses traversal agar tidak keluar dari batas array.

3. Apakah kegunaan dari method traverseInOrder()?

Method ini digunakan untuk **menelusuri dan menampilkan data dalam binary tree** secara **in-order traversal**, yaitu urutan:

1. Kiri,
2. Root (tengah),
3. Kanan.

Dalam konteks array:

- 2 * idxStart + 1 → anak kiri
- idxStart → node saat ini
- 2 * idxStart + 2 → anak kanan

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masing-masing?

Jika node berada di indeks 2, maka:

- Left child = 2 * 2 + 1 = 5
- Right child = 2 * 2 + 2 = 6

5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
idxLast = 6; digunakan untuk memberi tahu program bahwa **data mahasiswa hanya diisi sampai indeks ke-6** di array. Jadi saat traversal dilakukan, program hanya akan membaca sampai indeks ke-6 saja dan tidak melampaui data yang ada (menghindari error atau null pointer).
6. Mengapa indeks 2*idxStart+1 dan 2*idxStart+2 digunakan dalam pemanggilan rekursif, dan apa kaitannya dengan struktur pohon biner yang disusun dalam array?
Dalam **representasi binary tree menggunakan array**, ada aturan:

- **Indeks anak kiri** dari node ke-i = 2*i + 1
- **Indeks anak kanan** dari node ke-i = 2*i + 2

Ini merupakan **cara standar menyusun pohon biner dalam array** (mirip dengan struktur heap). Maka:

- Saat rekursi dilakukan, indeks ini digunakan untuk **menavigasi** dari satu node ke anak-anaknya.
- Tanpa pointer antar node, kita bisa tetap telusuri pohon hanya dengan menggunakan **indeks array** saja.

14.4 Tugas Praktikum

1. Buat method di dalam class BinaryTree00 yang akan menambahkan node dengan cara rekursif (addRekursif()).

```
168     void addRekursif(Mahasiswa11 data) {
169         root = addRekursif(root, data);
170     }
171
172     Node11 addRekursif(Node11 current, Mahasiswa11 data) {
173         if (current == null) {
174             return new Node11(data);
175         }
176         if (data.ipk < current.mahasiswa.ipk) {
177             current.left = addRekursif(current.left, data);
178         } else {
179             current.right = addRekursif(current.right, data);
180         }
181         return current;
182     }
```

2. Buat method di dalam class BinaryTree00 untuk menampilkan data mahasiswa dengan IPK paling kecil dan IPK yang paling besar (cariMinIPK() dan cariMaxIPK()) yang ada di dalam binary search tree.

```
184         public Mahasiswa11 cariMinIPK() {
185         if (root == null) return null;
186 
187         Node11 current = root;
188         while (current.left != null) {
189             current = current.left;
190         }
191         return current.mahasiswa;
192     }
193 
194     public Mahasiswa11 cariMaxIPK() {
195         if (root == null) return null;
196 
197         Node11 current = root;
198         while (current.right != null) {
199             current = current.right;
200         }
201         return current.mahasiswa;
202     }
```

3. Buat method dalam class BinaryTree00 untuk menampilkan data mahasiswa dengan IPK di atas suatu batas tertentu, misal di atas 3.50 (tampilMahasiswaIPKdiAtas(double ipkBatas)) yang ada di dalam binary search tree.

```
204     void tampilMahasiswaIPKdiAtas(double ipkBatas) {
205         tampilMahasiswaIPKdiAtas(root, ipkBatas);
206     }
207 
208     void tampilMahasiswaIPKdiAtas(Node11 node, double ipkBatas) {
209         if (node != null) {
210             tampilMahasiswaIPKdiAtas(node.left, ipkBatas);
211             if (node.mahasiswa.ipk > ipkBatas) {
212                 node.mahasiswa.tampilInformasi();
213             }
214             tampilMahasiswaIPKdiAtas(node.right, ipkBatas);
215         }
216     }
```

4. Modifikasi class BinaryTreeArray00 di atas, dan tambahkan :

- method add(Mahasiswa data) untuk memasukan data ke dalam binary tree
- method traversePreOrder()

```
26      public void add(Mahasiswa11 data) {
27      if (dataMahasiswa[0] == null) {
28          dataMahasiswa[0] = data;
29          idxLast = 0;
30          return;
31      }
32
33      int index = 0;
34      while (index < dataMahasiswa.length) {
35          if (data.ipk < dataMahasiswa[index].ipk) {
36              index = 2 * index + 1;
37          } else {
38              index = 2 * index + 2;
39          }
40
41          if (index < dataMahasiswa.length && dataMahasiswa[index] == null) {
42              dataMahasiswa[index] = data;
43              if (index > idxLast) {
44                  idxLast = index;
45              }
46              break;
47          }
48      }
49  }
50
51  public void traversePreOrder(int idxStart) {
52      if (idxStart <= idxLast) {
53          if (dataMahasiswa[idxStart] != null) {
54              dataMahasiswa[idxStart].tampilInformasi();
55              traversePreOrder(2 * idxStart + 1);
56              traversePreOrder(2 * idxStart + 2);
57          }
58      }
59  }
```