Nama : Dicky Darmawan

Kelas : TI 1B

NIM : 244107020037

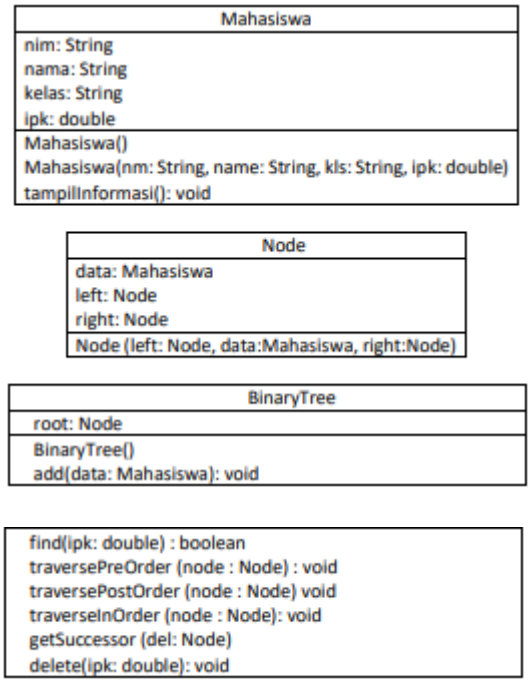
Kegiatan Praktikum 1: Implementasi Binary Search Tree menggunakan Linked List

Tahapan Pencobaan:

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
public class Mahasiswa08 {

    String nim, nama, kelas;
    double ipk;

    public Mahasiswa08() {

    }

    public Mahasiswa08(String nm, String name, String kls, double ipk) {
        nim = nm;
        nama = name;
        kelas = kls;
        this.ipk= ipk;
    }

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

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

```java
public class Node08 {

    Mahasiswa08 mahasiswa;
    Node08 left, right;

    public Node08() {

    }

    public Node08(Mahasiswa08 mahasiswa) {
        this.mahasiswa = mahasiswa;
        left = right = null;
    }
}
```

3. Di dalam class BinaryTree00, tambahkan atribut root.

```java
public class BinaryTree08 {

    Node08 root;
```

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

```java
public BinaryTree08() {
    root = null;
}

public boolean isEmpty() {
    return root == null;
}
```

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.

```
public void add(Mahasiswa08 mahasiswa) {
    Node08 newNode = new Node08(mahasiswa);
    if (isEmpty()) {
        root = newNode;
    } else {
        Node08 current = root;
        Node08 parent = null;
        while (true) {
            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;
                }
            }
        }
    }
}
```

6. Tambahkan method find()

```
public boolean find(double ipk) {
    boolean result = false;
    Node08 current = root;
    while (current != null) {
        if (current.mahasiswa.ipk == ipk) {
            result = true;
            break;
        } else if (ipk > current.mahasiswa.ipk) {
            current = current.right;
        } else {
            current = current.left;
        }
    }
    return result;
}
```

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

```
public void traversePreOrder(Node08 node) {
    if (node != null) {
        node.mahasiswa.tampilInformasi();
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

public void traversePostOrder(Node08 node) {
    if (node!=null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        node.mahasiswa.tampilInformasi();
    }
}

public void traverseInOrder(Node08 node) {
    if (node!= null) {
        traverseInOrder(node.left);
        node.mahasiswa.tampilInformasi();
        traverseInOrder(node.right);
    }
}
```

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

```
Node08 getSuccesor(Node08 del) {
    Node08 successor = del.right;
    Node08 successorParent = del;
    while (successor.left != null) {
        successorParent = successor;
        successor = successor.left;
    }
    if (successor != del.right) {
        successorParent.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}
```

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

```java
public void delete(double ipk) {
    if (isEmpty()) {
        System.out.println(x:"Binary tree kosong");
        return;
    }
    // cari node (current) yang akan dihapus
    Node08 parent = root;
    Node08 current = root;
    boolean isLeftChild = false;
    while (current != null) {
        if (current.mahasiswa.ipk == ipk) {
            break;
        } else if (ipk < current.mahasiswa.ipk) {
            parent = current;
            current = current.left;
            isLeftChild = true;
        } else if (ipk > current.mahasiswa.ipk) {
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
}
```

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

```java
// Penghapusan
if (current == null) {
    System.out.println(x:"Data tidak ditemukan");
    return;
} else {
    //Jika tidak ada anak (leaf), maka node dihapus
    if (current.left == null && current.right == null ) {
        if (current == root) {
            root = null;
        } else {
            if (isLeftChild) {
                parent.left = null;
            } else{
                parent.right = null;
            }
        }
    } else if (current.left == null) {// Jika 1 anak di kanan
        if (current == root) {
        root = current.right;
        } else {
            if (isLeftChild) {
                parent.left = current.right;
            } else {
                parent.right = current.right;
            }
        }
    } else if (current.right == null) {// Jika 1 anak di kiri
        if (current == root) {
            root = current.left;
        } else {
            if(isLeftChild) {
                parent.left=current.left;
            } else {
               parent.right = current.left;
            }
        }
```

```
} else { // Jika punya 2 anak
    Node08 successor = getSuccesor(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.left = successor;
        }
    }
    successor.left = current.left;
}
```

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

```
public class BinaryTreeMain08 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTree08 bst = new BinaryTree08();

        bst.add(new Mahasiswa08(nm:"244107024", name:"Ali", kls:"A", ipk:3.57));
        bst.add(new Mahasiswa08(nm:"244107025", name:"Bahlil", kls:"B", ipk:3.85));
        bst.add(new Mahasiswa08(nm:"244107026", name:"Cahyo", kls:"C", ipk:3.21));
        bst.add(new Mahasiswa08(nm:"244107027", name:"Damar", kls:"D", ipk:3.54));

        System.out.println(x:"\nDaftar semua mahasiswa (in order traversal):");
        bst.traverseInOrder(bst.root);

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

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

        bst.add(new Mahasiswa08(nm:"244107028", name:"Harry", kls:"A", ipk:3.72));
        bst.add(new Mahasiswa08(nm:"244107029", name:"Ron", kls:"B", ipk:3.37));
        bst.add(new Mahasiswa08(nm:"244107030", name:"Malfoy", kls:"C", ipk:3.46));
        System.out.println(x:"\nDaftar semua mahasiswa setelah penambahan 3 mahasiswa:");
        System.out.println(x:"InOrder Traversal:");
        bst.traverseInOrder(bst.root);
        System.out.println(x:"\nPreOder Traversal");
        bst.traversePreOrder(bst.root);
        System.out.println(x:"\nPostOrder Traversal");
        bst.traversePostOrder(bst.root);

        System.out.println(x:"\nPenghapusan data mahasiswa");
        bst.delete(ipk:3.57);
        System.out.println(x:"\nDaftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):");
        bst.traverseInOrder(bst.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 order traversal):
NIM: 244107026 Nama: Cahyo Kelas: C IPK: 3.21
NIM: 244107027 Nama: Damar Kelas: D IPK: 3.54
NIM: 244107024 Nama: Ali Kelas: A IPK: 3.57
NIM: 244107025 Nama: Bahlil 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: 244107026 Nama: Cahyo Kelas: C IPK: 3.21
NIM: 244107029 Nama: Ron Kelas: B IPK: 3.37
NIM: 244107030 Nama: Malfoy Kelas: C IPK: 3.46
NIM: 244107027 Nama: Damar Kelas: D IPK: 3.54
NIM: 244107024 Nama: Ali Kelas: A IPK: 3.57
NIM: 244107028 Nama: Harry Kelas: A IPK: 3.72
NIM: 244107025 Nama: Bahlil Kelas: B IPK: 3.85

PreOder Traversal
NIM: 244107024 Nama: Ali Kelas: A IPK: 3.57
NIM: 244107026 Nama: Cahyo Kelas: C IPK: 3.21
NIM: 244107027 Nama: Damar Kelas: D IPK: 3.54
NIM: 244107029 Nama: Ron Kelas: B IPK: 3.37
NIM: 244107030 Nama: Malfoy Kelas: C IPK: 3.46
NIM: 244107025 Nama: Bahlil Kelas: B IPK: 3.85
NIM: 244107028 Nama: Harry Kelas: A IPK: 3.72

PostOrder Traversal
NIM: 244107030 Nama: Malfoy Kelas: C IPK: 3.46
NIM: 244107029 Nama: Ron Kelas: B IPK: 3.37
NIM: 244107027 Nama: Damar Kelas: D IPK: 3.54
NIM: 244107026 Nama: Cahyo Kelas: C IPK: 3.21
NIM: 244107028 Nama: Harry Kelas: A IPK: 3.72
NIM: 244107025 Nama: Bahlil Kelas: B IPK: 3.85
NIM: 244107024 Nama: Ali Kelas: A IPK: 3.57
```

```
Penghapusan data mahasiswa
Jika 2 anak, current =
NIM: 244107028 Nama: Harry Kelas: A IPK: 3.72

Daftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):
NIM: 244107026 Nama: Cahyo Kelas: C IPK: 3.21
NIM: 244107029 Nama: Ron Kelas: B IPK: 3.37
NIM: 244107030 Nama: Malfoy Kelas: C IPK: 3.46
NIM: 244107027 Nama: Damar Kelas: D IPK: 3.54
NIM: 244107028 Nama: Harry Kelas: A IPK: 3.72
NIM: 244107025 Nama: Bahlil Kelas: B IPK: 3.85
```

Pertanyaan

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

   Karena data disimpan secara terurut. Jadi kalau kita nyari sesuatu, kita bisa langsung milih ke kiri atau ke kanan tanpa harus cek semua node, beda dengan binary tree biasa yang nggak ada urutannya.
2. Untuk apakah di class Node, kegunaan dari atribut left dan right?

   dipakai buat nunjuk ke anak kiri dan anak kanan dari node tersebut

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

      Untuk menunjuk node awa; dari seluruh tree, jadi sebagai awalan untuk traverse

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

      Masih kosong atau null
3. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

   Jika tree masih kosong dan mau ditambahin node baru, maka node pertama yang dimasukkan adalah root
4. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

```
parent = current;                Mahasiswa08 mah
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;
    }
}
```

   Melakukan pengecekan apakah data IPK mahasiswa lebih kecil dari node yang current. Jika iya, dia lanjut ke anak eft, dan jika anak kirinya null, node baru langsung ditambahkan di situ. Jika tidak, akan pindah ke anak right dan logika yang sama akan dijalankan. Jadi ini proses tersebut bertujuan untuk mencari tempat yang pas untuk menambahkan data
5. Jelaskan langkah-langkah pada method delete() saat menghapus sebuah node yang memiliki dua anak. Bagaimana method getSuccessor() membantu dalam proses ini?

Saat method delete() dijalankan, langkah-langkahnya tergantung node yang akan dihapus. Jika node tidak memiliki anak, maka akan langsung dihapus. Jika punya satu anak, anaknya langsung menggantikan posisi node itu. Tapi jika punya dua anak, perlu dicari node pengganti yang pas, yaitu getSuccessor(). Method ini membantu mencari node yang nilainya paling kecil di subtree kanan, lalu akan dijadikan pengganti node yang dihapus supaya struktur tree tetap rapi.

Kegiatan Praktikum 2: Implementasi Binary Tree dengan Array

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().

```
public class BinaryTreeArray08 {
    Mahasiswa08[] dataMahasiswa;
    int idxLast;

    public BinaryTreeArray08() {
        this.dataMahasiswa = new Mahasiswa08[10];
    }

    public void populateData (Mahasiswa08 dataMhs[], int idxLast) {
        this.dataMahasiswa = dataMhs;
        this.idxLast = idxLast;
    }

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

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

```
public class BinaryTreeArrayMain08 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray08 bta = new BinaryTreeArray08();
        Mahasiswa08 mhs1 = new Mahasiswa08(nm:"244107024", name:"Ali", kls:"A", ipk:3.57);
        Mahasiswa08 mhs2 = new Mahasiswa08(nm:"244107025", name:"Bahlil", kls:"B", ipk:3.85);
        Mahasiswa08 mhs3 = new Mahasiswa08(nm:"244107026", name:"Cahyo", kls:"C", ipk:3.21);
        Mahasiswa08 mhs4 = new Mahasiswa08(nm:"244107027", name:"Damar", kls:"D", ipk:3.54);

        Mahasiswa08 mhs5 = new Mahasiswa08(nm:"244107028", name:"Harry", kls:"A", ipk:3.72);
        Mahasiswa08 mhs6 = new Mahasiswa08(nm:"244107029", name:"Ron", kls:"B", ipk:3.37);
        Mahasiswa08 mhs7 = new Mahasiswa08(nm:"244107030", name:"Malfoy", kls:"C", ipk:3.46);

        Mahasiswa08[] 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: 244107027 Nama: Damar Kelas: D IPK: 3.54
NIM: 244107025 Nama: Bahlil Kelas: B IPK: 3.85
NIM: 244107027 Nama: Damar Kelas: D IPK: 3.54
NIM: 244107024 Nama: Ali Kelas: A IPK: 3.57
NIM: 244107027 Nama: Damar Kelas: D IPK: 3.54
NIM: 244107025 Nama: Bahlil Kelas: B IPK: 3.85
NIM: 244107027 Nama: Damar Kelas: D IPK: 3.54
```

Pertanyaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?
   Data menyimpan elemen-elemen tree dalam bentuk array. idxLast menunjukkan indeks terakhir yang terisi, berguna untuk membatasi proses pada data yang valid saja.
2. Apakah kegunaan dari method populateData()?
   Berfungsi mengisi array data dengan nilai-nilai node dan menetapkan nilai idxLast sesuai jumlah elemen.
3. Apakah kegunaan dari method traverseInOrder()?
   Melakukan penelusuran secara inorder: anak kiri - root - anak kanan, sehingga data ditampilkan secara terurut.
4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masing-masing?
   Anak kiri berada di indeks 5, anak kanan di indeks 6 (2*2+1 dan 2*2+2).
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   Menandakan data terakhir yang valid berada di indeks ke-6, sebagai batas proses.
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?
   Karena dalam array, struktur pohon biner disusun agar anak kiri dan kanan dapat dihitung langsung dari indeks induknya menggunakan rumus tersebut.

Tugas Praktikum

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

```
public void addRekursif(Mahasiswa08 data) {
    root = addRekursif(root, data);
}

public Node08 addRekursif(Node08 current, Mahasiswa08 data) {
    if (current == null) {
        return new Node08(data);
    }

    if (data.ipk < current.mahasiswa.ipk) {
        current.left = addRekursif(current.left, data);
    } else {
        current.right = addRekursif(current.right, data);
    }

    return current;
}
```

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.

```
public Mahasiswa08 cariMinIPK() {
    if (root == null) return null;
    Node08 current = root;
    while (current.left != null) {
        current = current.left;
    }
    return current.mahasiswa;
}

public Mahasiswa08 cariMaxIPK() {
    if (root == null) return null;
    Node08 current = root;
    while (current.right != null) {
        current = current.right;
    }
    return current.mahasiswa;
}
```

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.

```java
public void tampilMahasiswaIPKdiAtas(double ipkBatas) {
    tampilMahasiswaIPKdiAtas(root, ipkBatas);
}

public void tampilMahasiswaIPKdiAtas(Node08 node, double ipkBatas) {
    if (node != null) {
        tampilMahasiswaIPKdiAtas(node.left, ipkBatas);
        if (node.mahasiswa.ipk > ipkBatas) {
            System.out.println("Nama: " + node.mahasiswa.nama + ", IPK: " + node.mahasiswa.ipk);
        }
        tampilMahasiswaIPKdiAtas(node.right, ipkBatas);
    }
}
```

4. Modifikasi class BinaryTreeArray00 di atas, dan tambahkan :

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

```java
public void add(Mahasiswa08 dataBaru) {
    if (idxLast >= dataMahasiswa.length - 1) {
        System.out.println(x:"Tree penuh");
        return;
    }
    idxLast++;
    dataMahasiswa[idxLast] = dataBaru;
}
```

- method traversePreOrder()

```java
public void traversePreOrder() {
    traversePreOrder(idx:0);
}

public void traversePreOrder(int idx) {
    if (idx > idxLast || dataMahasiswa[idx] == null) return;
    System.out.println("Nama: " + dataMahasiswa[idx].nama + ", IPK: " + dataMahasiswa[idx].ipk);
    traversePreOrder(2 * idx + 1); // anak kiri
    traversePreOrder(2 * idx + 2); // anak kanan
}
```