Nama : Wahyudi Satriawan Hamid

Nim : 244107020137

Verifikasi Hasil Percobaan 1

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

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
```

Pertanyaan 1

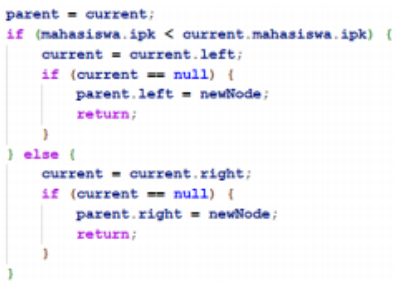
1. Karena binary search tree (BST) menyimpan data secara terurut:
   - Nilai lebih kecil ada di kiri
   - Nilai lebih besar ada di kanan

     Ini membuat proses pencarian lebih cepat karena bisa mengabaikan separuh subtree di setiap langkah (mirip seperti binary search pada array).
2. Atribut left menunjuk ke anak kiri, dan right menunjuk ke anak kanan.

   Mereka menyusun struktur pohon biner dan digunakan untuk traversal, pencarian, dan operasi lain.

3. A) root adalah node utama (paling atas) dari sebuah pohon. Semua proses pencarian, penambahan, dan penghapusan data dimulai dari node ini.
   B) null, karena pohon masih kosong dan belum ada node yang ditambahkan.

4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   - Node baru akan menjadi root
   - Karena `root == null`, maka node tersebut langsung diset sebagai akar pohon

5. 

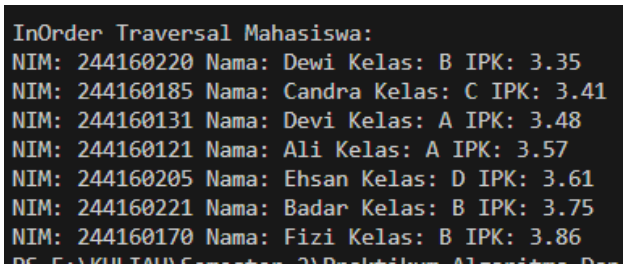
```
parent = current;
if (mahasiswa.ipk < current.mahasiswa.ipk) {
    current = current.left;
    if (current == null) {
        parent.left = newNode;
        return;
    }
} else {
    current = current.right;
    if (current == null) {
        parent.right = newNode;
        return;
    }
}
```

   Kode ini adalah bagian dari logika penyisipan (insert) BST:
   - parent = current; → menyimpan referensi parent node sebelum turun ke kiri/kanan
   - if (mahasiswa.ipk < current.ipk) → cek apakah harus masuk ke subtree kiri
   - if (current == null) → jika posisi kosong ditemukan, node baru disisipkan di situ
   - parent.left = newNode atau parent.right = newNode → sambungkan node baru ke parent-nya

6. Jelaskan langkah-langkah pada method delete() saat menghapus sebuah node yang memiliki dua anak. Bagaimana method getSuccessor() membantu dalam proses ini?
   - Saat node yang akan dihapus punya dua anak, kita perlu mengganti node tersebut dengan data pengganti (successor).
   - `getSuccessor()` akan mencari node terkecil di subtree kanan dari node yang dihapus (yaitu node paling kiri di kanan).
   - Langkah:
     a. Cari successor dengan `getSuccessor()`
     b. Salin data successor ke node yang dihapus
     c. Hapus node successor asli dari posisi aslinya

   Metode ini mempertahankan sifat BST setelah penghapusan.

Verifikasi Hasil Percobaan 2



Pertanyaan 2

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?
   - `data` (atau `dataMahasiswa`) menyimpan array dari node (dalam hal ini, objek `Mahasiswa25`) sebagai representasi pohon.
   - `idxLast` menyimpan indeks terakhir yang terisi dalam array, agar traversal tidak melewati batas array yang kosong.

2. Apakah kegunaan dari method populateData()?
   Untuk mengisi array pohon (dataMahasiswa) dengan data yang sudah ditentukan, dan mengatur batas akhir data (idxLast) untuk traversal.

3. Apakah kegunaan dari method traverseInOrder()?
   Untuk melakukan traversal in-order secara rekursif pada array pohon:
   - Mengunjungi subtree kiri
   - Mencetak node sekarang
   - Mengunjungi subtree kanan

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan right child masing-masing?
   - 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?
   Untuk menandai bahwa data mahasiswa hanya ada sampai indeks ke-6 dalam array. Ini penting agar traversal berhenti di situ dan tidak membaca elemen null setelahnya.

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 representasi binary tree berbasis array:
   - Jika parent ada di indeks i
     - Maka left child ada di 2*i + 1
     - Dan right child ada di 2*i + 2

   Ini adalah rumus standar untuk mengakses anak dari parent dalam struktur pohon yang disimpan sebagai array (seperti pada heap atau binary tree array).

Tugas

1.
```
public void addRekursif(Mahasiswa25 data) {
    root = tambahRekursif(root, data);
}

public Node25 tambahRekursif(Node25 current, Mahasiswa25 data) {
    if (current == null) {
        return new Node25(data);
    }

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

    return current;
}
```

2.
```
public void cariMinIPK() {
    if (isEmpty()) {
        System.out.println("Tree kosong");
        return;
    }

    Node25 current = root;
    while (current.left != null) {
        current = current.left;
    }
    System.out.println("Mahasiswa dengan IPK terkecil:");
    current.mahasiswa.tampilInformasi();
}

public void cariMaxIPK() {
    if (isEmpty()) {
        System.out.println("Tree kosong");
        return;
    }

    Node25 current = root;
    while (current.right != null) {
        current = current.right;
    }
    System.out.println("Mahasiswa dengan IPK terbesar:");
    current.mahasiswa.tampilInformasi();
}
```

3.
```
public void tampilMahasiswaIPKdiAtas(double ipkBatas) {
    tampilMahasiswaIPKdiAtasRecursive(root, ipkBatas);
}

private void tampilMahasiswaIPKdiAtasRecursive(Node25 node, double ipkBatas) {
    if (node != null) {
        tampilMahasiswaIPKdiAtasRecursive(node.left, ipkBatas);
        if (node.mahasiswa.ipk > ipkBatas) {
            node.mahasiswa.tampilInformasi();
        }
        tampilMahasiswaIPKdiAtasRecursive(node.right, ipkBatas);
    }
}
```

4.
```
public void add(Mahasiswa25 data) {
    for (int i = 0; i < dataMahasiswa.length; i++) {
        if (dataMahasiswa[i] == null) {
            dataMahasiswa[i] = data;
            idxLast = i;
            return;
        }
    }
    System.out.println("Tree penuh, tidak bisa menambahkan data.");
}

public void traversePreOrder(int
    if (idxStart <= idxLast && d
        dataMahasiswa[idxStart].tampilInformasi();
        traversePreOrder(2 * idxStart + 1); // kiri
        traversePreOrder(2 * idxStart + 2); // kanan
    }
}
```
void Jobsheet14.Mahasiswa25.tampilInformasi()