Nama : Nadhya Sigalingging

Kelas : 47-02

NIM : 607062300065

Penjelasan jurnal 13

1. Kelas BST

```
public class BST<E extends Comparable<E>> {
    public BSTNode<E> root;

    public BST() {
        root = null;
    }
}
```

- **Kelas BST** adalah kelas generik yang dapat digunakan dengan tipe data apapun yang mengimplementasikan antarmuka Comparable.
- Atribut root adalah node akar dari pohon.
- Konstruktor BST() menginisialisasi root menjadi null, menandakan pohon awalnya kosong.

```
public void insert(E data) {
    if (root == null) {
        root = new BSTNode<E>(data);
    } else {
        root.insert(data);
    }
}
```

- Metode insert digunakan untuk menambahkan elemen baru ke dalam pohon.
- Jika root kosong (null), elemen baru menjadi root.
- Jika root tidak kosong, elemen baru disisipkan pada posisi yang sesuai dengan bantuan metode insert dari kelas BSTNode.

```
•   public void search(E val) {
•       E hasil = searchBSTHelper(root, val);
•       if (hasil != null) {
•           System.out.println("Karakter " + hasil + " ada didalam
    program");
•       } else {
```

```
            System.out.println("Karakter " + val + " tidak ada didalam
    program");
        }
    }
```

- Metode search digunakan untuk mencari elemen dalam BST.
- Menggunakan metode bantu searchBSTHelper untuk melakukan pencarian.
- Menampilkan pesan apakah elemen ditemukan atau tidak.

```
public E searchBSTHelper(BSTNode<E> node, E val) {
    E result = null;
    if (node != null) {
        if (val.equals(node.getData())) {
            result = node.getData();
        } else if (val.compareTo(node.getData()) < 0) {
            result = searchBSTHelper(node.getLeftNode(), val);
        } else {
            result = searchBSTHelper(node.getRightNode(), val);
        }
    }
    return result;
}
```

- Metode searchBSTHelper mencari elemen secara rekursif.
- Jika node saat ini tidak null, periksa apakah elemen yang dicari sama dengan data pada node saat ini.
- Jika lebih kecil, lanjutkan pencarian di subtree kiri; jika lebih besar, lanjutkan di subtree kanan.
- Jika elemen ditemukan, kembalikan nilai elemen tersebut.

```
public boolean BSTsearch(E val) {
    return search(root, val);
}

private boolean search(BSTNode<E> r, E val) {
    if (r.getData().equals(val)) {
        return true;
    }
    if (r.getLeftNode() != null && search(r.getLeftNode(), val)) {
        return true;
    }
    if (r.getRightNode() != null && search(r.getRightNode(), val)) {
```

```
        return true;
    }
    return false;
}
```

- Metode BSTsearch mengembalikan true jika elemen ditemukan, false jika tidak.
- Metode ini mencari elemen mulai dari root, melanjutkan ke subtree kiri dan kanan sesuai kebutuhan.

```
public void inorder() {
    inorder(root);
}

private void inorder(BSTNode<E> r) {
    if (r != null) {
        inorder(r.getLeftNode());
        System.out.print(r.getData() + " ");
        inorder(r.getRightNode());
    }
}

public void preorder() {
    preorder(root);
}

private void preorder(BSTNode<E> r) {
    if (r != null) {
        System.out.print(r.getData() + " ");
        preorder(r.getLeftNode());
        preorder(r.getRightNode());
    }
}

public void postorder() {
    postorder(root);
}

private void postorder(BSTNode<E> r) {
    if (r != null) {
        postorder(r.getLeftNode());
        postorder(r.getRightNode());
        System.out.print(r.getData() + " ");
    }
}
```

- **Metode inorder:**
  Mengunjungi subtree kiri, node saat ini, kemudian subtree kanan.

- **Metode preorder:**
  Mengunjungi node saat ini, subtree kiri, kemudian subtree kanan.

- **Metode postorder:**
  Mengunjungi subtree kiri, subtree kanan, kemudian node saat ini.

2. Kelas BSTNode

```
public class BSTNode<E extends Comparable<E>> {
    public BSTNode<E> leftNode;
    public E data;
    public BSTNode<E> rightNode;

    public BSTNode(E item) {
        data = item;
        leftNode = rightNode = null;
    }

    public E getData() {
        return data;
    }

    public BSTNode<E> getLeftNode() {
        return leftNode;
    }

    public BSTNode<E> getRightNode() {
        return rightNode;
    }

    public void insert(E insertValue) {
        if (insertValue.compareTo(data) < 0) {
            if (leftNode == null) {
                leftNode = new BSTNode<E>(insertValue);
            } else {
                leftNode.insert(insertValue);
            }
        } else if (insertValue.compareTo(data) > 0) {
            if (rightNode == null) {
                rightNode = new BSTNode<E>(insertValue);
            } else {
                rightNode.insert(insertValue);
```

```
            }
        }
    }
}
```

- **Konstruktor BSTNode(E item):**

✓ Menyimpan nilai item dalam data.
✓ Menginisialisasi leftNode dan rightNode menjadi null.

- **Metode getData():**

✓ Mengembalikan data dari node saat ini.

- **Metode getLeftNode() dan getRightNode():**

✓ Mengembalikan node anak kiri dan kanan.

- **Metode insert(E insertValue):**

✓ Menyisipkan nilai dengan tepat sesuai aturan BST.
✓ Jika nilai lebih kecil dari data saat ini, disisipkan ke subtree kiri; jika lebih besar, ke subtree kanan.

3. Kelas Main

```
4. public class Main {
5.     public static void main(String[] args) {
6.         BST<Character> bst = new BST<>();
7.         bst.insert('F');
8.         bst.insert('E');
9.         bst.insert('D');
10.        bst.insert('C');
11.        bst.insert('B');
12.        bst.insert('H');
13.        bst.insert('G');
14.        bst.insert('K');
15.        bst.insert('J');
16.
17.        System.out.print("\nPost order: ");
18.        bst.postorder();
19.        System.out.print("\nPre order: ");
20.        bst.preorder();
21.        System.out.print("\nIn order: ");
22.        bst.inorder();
23.        System.out.print("\nKarakter: ");
24.        bst.search('K');
25.        bst.search('A');
26.    }
```

```
27.}
28.
```

- **Bagian Main**

✓ Membuat instance BST untuk menyimpan karakter.
✓ Menyisipkan karakter-karakter ke dalam pohon dengan urutan tertentu.
✓ Melakukan traversal post-order, pre-order, dan in-order, serta mencari karakter K dan A.

- **Hasil:**

✓ Menampilkan urutan traversal dan hasil pencarian.
✓ K ditemukan, sedangkan A tidak ditemukan karena tidak ada dalam elemen yang disisipkan.

- **Hasil Program Ketika Dijalankan**

```
Post order : B C D E G J K H F
Pre order : F E D C B H G K J
In order : B C D E F G H J K
Karakter: Karakter K ada didalam program
Karakter A tidak terdapat didalam program
```

- **Gambar pohon**