Tree.java

```java
public class Tree<E extends Comparable<E>> {
    private TreeNode<E> root;

    public Tree() {
        root = null;
    }

```

- Deklarasi Generik: E adalah tipe parameter yang harus Comparable<E>. Ini berarti bahwa elemen yang disimpan dalam pohon harus dapat dibandingkan satu sama lain.
- root: Merupakan referensi ke node akar dari pohon.
- Konstruktor: Menginisialisasi root sebagai null, yang berarti pohon dimulai dalam keadaan kosong.

```java
    public void insertNode(E insertValue) {
        System.out.print(insertValue + " ");
        if (root == null) {
            root = new TreeNode<E>(insertValue);
        } else {
            root.insert(insertValue);
        }
    }

```

- insertNode(E insertValue): Menyisipkan nilai baru ke dalam pohon.
- Jika root kosong (pohon masih kosong), maka node baru dengan nilai insertValue menjadi akar.
- Jika root tidak kosong, delegasikan penyisipan ke metode insert dari TreeNode.

```java
    private void preorderHelper(TreeNode<E> node) {
        if (node == null) {
            return;
        }

        System.out.printf("%s ", node.getData());
        preorderHelper(node.getLeftNode());
        preorderHelper(node.getRightNode());
    }

    public void preorderTraversal() {
        preorderHelper(root);
    }

```

- preorderHelper(TreeNode<E> node): Metode rekursif untuk traversal preorder.

- Kunjungi node saat ini, kemudian rekursif kunjungi left child dan right child.
- preorderTraversal(): Memulai traversal preorder dari root.

```
31      private void inorderHelper(TreeNode<E> node) {
32          if (node == null) {
33              return;
34          }
35
36          inorderHelper(node.getLeftNode());
37          System.out.printf("%s ", node.getData());
38          inorderHelper(node.getRightNode());
39      }
40
41      public void inorderTraversal() {
42          inorderHelper(root);
43      }
44
```

- inorderHelper(TreeNode<E> node): Metode rekursif untuk traversal inorder.
- Rekursif kunjungi left child, kemudian node saat ini, dan kemudian right child.
- inorderTraversal(): Memulai traversal inorder dari root.

```
45      private void postorderHelper(TreeNode<E> node) {
46          if (node == null) {
47              return;
48          }
49
50          postorderHelper(node.getLeftNode());
51          postorderHelper(node.getRightNode());
52          System.out.printf("%s ", node.getData());
53      }
54
55      public void postorderTraversal() {
56          postorderHelper(root);
57      }
58
```

- postorderHelper(TreeNode<E> node): Metode rekursif untuk traversal postorder.
- Rekursif kunjungi left child, kemudian right child, dan terakhir node saat ini.
- postorderTraversal(): Memulai traversal postorder dari root.

```java
    private boolean searchBSTHelper(TreeNode<E> node, E key) {
        boolean result = false;

        if (node != null) {
            if (key.equals(node.getData())) {
                result = true;
            } else if (key.compareTo(node.getData()) < 0) {
                result = searchBSTHelper(node.getLeftNode(), key);
            } else {
                result = searchBSTHelper(node.getRightNode(), key);
            }
        }
        return result;
    }

    public void searchBST(E key) {
        boolean hasil = searchBSTHelper(root, key);

        if (hasil) {
            System.out.println("Data " + key + " ditemukan pada tree");
        } else {
            System.out.println("Data "+ key + " tidak ditemukan pada tree");
        }
    }


}
```

- searchBSTHelper(TreeNode<E> node, E key): Metode rekursif untuk mencari key dalam pohon.
- Jika node saat ini null, kembalikan false.
- Jika key sama dengan data pada node saat ini, kembalikan true.
- Jika key lebih kecil, rekursif cari di left child.
- Jika key lebih besar, rekursif cari di right child.
- searchBST(E key): Memulai pencarian key dari root dan mencetak hasilnya.

TreeNode.java

```java
public class TreeNode<E extends Comparable<E>> {
```

- E harus mengimplementasikan antarmuka Comparable<E>, artinya objek dari tipe E dapat dibandingkan satu sama lain.

```java
    public TreeNode(E nodeData) {
        data = nodeData;
        leftNode = rightNode = null;
    }
```

- Konstruktor yang menerima satu parameter nodeData dan menginisialisasi data node tersebut.
- leftNode dan rightNode diinisialisasi menjadi null.

```
24      public void insert(E insertValue) {
25          if (insertValue.compareTo(data) < 0) {
26              if (leftNode == null) {
27                  leftNode = new TreeNode<E>(insertValue);
28              } else {
29                  leftNode.insert(insertValue);
30              }
31          }
```

- Jika leftNode adalah null, buat node anak kiri baru dengan insertValue.
- Jika tidak, lakukan rekursi dengan memanggil metode insert pada leftNode.

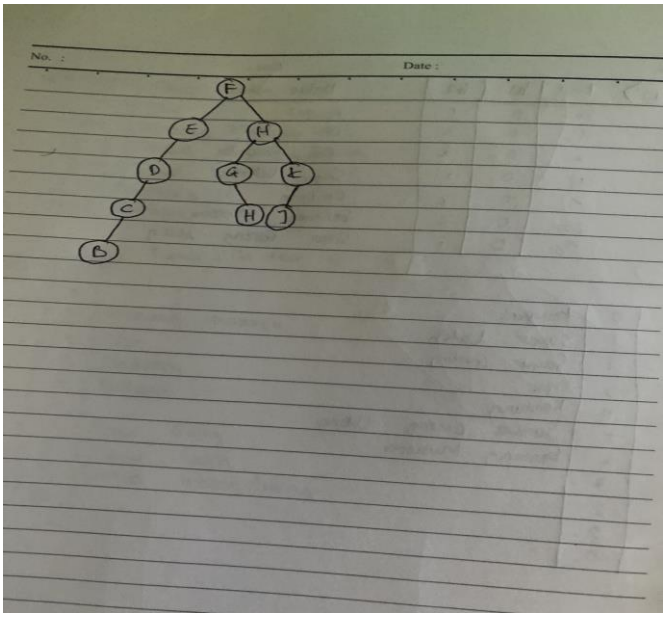
```
32          else if (insertValue.compareTo(data) > 0) {
33              if (rightNode == null) {
34                  rightNode = new TreeNode<E>(insertValue);
35              } else {
36                  rightNode.insert(insertValue);
37              }
38          }
```

- Jika rightNode adalah null, buat node anak kanan baru dengan insertValue.
- Jika tidak, lakukan rekursi dengan memanggil metode insert pada rightNode.

```
40          else {
41              if (leftNode == null) {
42                  leftNode = new TreeNode<E>(insertValue);
43              } else {
44                  leftNode.insert(insertValue);
45              }
46          }
47      }
48
49      @Override
50      public String toString() {
51          return "TreeNode [leftNode=" + leftNode + ", data=" + data + ", rightNode=" + rightNode + "]";
52      }
53
54
55
56  }
57
```

- Jika insertValue sama dengan data (situasi ini biasanya tidak terjadi dalam tree biner pencarian yang khas karena tidak mendukung duplikat), dalam implementasi ini kita memperlakukan duplikat dengan memasukkan ke anak kiri.

- Mengoverride metode toString dari kelas Object untuk memberikan representasi string dari node, termasuk anak kiri, data, dan anak kanan.



Pertama-tama kita memasukkan huruf F yang dijadikan sebagai root. Lalu dilanjutkan oleh huruf E yang diletakkan di sebelah kiri karena nilainya lebih kecil dari F. Nilai karakter ditentukan oleh urutan alfabet dimulai dari A dengan nilai paling kecil dan Z dengan nilai paling besar. Setelah itu masuk huruf H yang diletakkan disebelah kanan F. Proses ini dilanjut sampai urutan selesai

Dapat dilihat bahwa huruf H dimasukkan dua kali. Pada kodingan ini tidak melakukan penangan untuk nilai yang sama dengan data yang akan dimasukkan, sehingga huruf H tidak dimasukkan kedalam tree. Tetapi saya memasukkan untuk penulisan di Kertas