Panduan untuk implementasi **Binary Search Tree**, dan beberapa operasinya. Ingat, yang diberikan di sini hanyalah salah satu cara implementasi saja.

```
//BinaryNode
public class BinaryNode<E> {

    E element;
    BinaryNode<E> left;
    BinaryNode<E> right;


    public BinaryNode (E element, BinaryNode<E> left, BinaryNode<E> right) {
        this.element = element;
        this.left = left;
        this.right = right;
    }


    public BinaryNode (E element) {
        this (element, null, null);
    }


    void printInOrder () {  //lengkapi sekali lagi









    }
}
```

```
//BST
public class BST<E extends Comparable<? super E>> {  //mengapa ?

    BinaryNode<E> root;

    public void printInOrder () {
        if (root != null) {
            root.printInOrder ();
            System.out.println ();
        }
    }

    public void insert (E x) {
        root = insert (x, root);
    }
```

```
    //insert sebuah elemen ke BST, root dari t
    private BinaryNode<E> insert (E x, BinaryNode<E> t) {
        if (t == null) {
            t = new BinaryNode<E> (x, null, null);
        } else if (x.compareTo (t.element) < 0) {
            t.left = insert (x, t.left);
        } else if (x.compareTo (t.element) > 0) {
            t.right = insert (x, t.right);
        } else {
            throw new IllegalArgumentException ("Duplicate item");
        }
        return t;
    }

    //mengembalikan node yang mengandung elemen paling kecil (iterasi)
    //dari BST, root t
    public BinaryNode<E> findMin (BinaryNode<E> t) {
        if (t == null) {
            throw new IllegalArgumentException ("Empty tree");
        }









    }

    //mengembalikan node yang mengandung elemen paling kecil (rekursif)
    //dari BST, root t
    public BinaryNode<E> findMinRec (BinaryNode<E> t) {
         if (t == null) {
              throw new IllegalArgumentException("Empty tree");
         }









    }
```

```
//mengembalikan elemen yang paling kecil, dari root
//- gunakan iterasi
public E findMin () {
    if (root == null) {
        throw new IllegalArgumentException ("Empty tree");
    }

    BinaryNode<E> t = root;









}


//mencari value di BST, bila ada kembalikan alamat Node tersebut
public BinaryNode<E> find (E value) {  //gunakan iterasi
    BinaryNode<E> t = root;

    while (t != null) {









    }

    return null;
}



public BinaryNode<E> findRec (E value) {
    return findRec (value, root);
}
```

```
//mencari value di BST, bila ada kembalikan alamat Node tersebut
private BinaryNode<E> findRec (E value, BinaryNode<E> t) { //rekursif
    if (t == null) {
        return null;
    } else {









    }
}


//hapus Node yang mengandung elemen terkecil pada BST
public void removeMin () {
    if (root == null) {
        throw new IllegalArgumentException ("empty tree");
    }
    root = removeMin (root);
}



//hapus Node yang mengandung elemen terkecil pada BST, root dari t
private BinaryNode<E> removeMin (BinaryNode<E> t) {




}

//hapus Node yang mengandung elemen terbesar pada BST, root dari t
private BinaryNode<E> removeMax (BinaryNode<E> t) {




}
```

```
    public void remove (E x) {
        if (root == null) {
            throw new IllegalArgumentException ("empty tree");
        }
        root = remove (x, root);
    }

    //hapus x dari BST yang root-nya adalah t
    private BinaryNode<E> remove (E x, BinaryNode<E> t) {
        if (t == null) {
            return null;
        }









    }

}
```

Testing

```
public class BSTTest
{
    public static void main (String[] args)
    {
        BST<Integer> tree = new BST<Integer> ();

        tree.printInOrder ();

        tree.insert (6);
        tree.insert (3);
        tree.insert (8);
        tree.insert (5);
        tree.insert (15);

        tree.printInOrder ();


        System.out.println (tree.findMin(tree.root).element);
        System.out.println (tree.findMinRec(tree.root).element);
        tree.remove (10);
        tree.printInOrder ();
```

```
        tree.remove (6);
        tree.printInOrder ();
        tree.insert (2);
        tree.insert (4);
        tree.printInOrder ();
        tree.remove (3);
        tree.printInOrder ();

    }
}
```