JAVA program to perform Insertion. Deletion, Find min, Find max, Find successor and Find predecessor in BST.

```
class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
    }
}

class BST {
    Node root;

    void insert(int key) {
        root = insertRec(root, key);
    }

    Node insertRec(Node root, int key) {
        if (root == null) {
            return new Node(key);
        }
        if (key < root.key) {
            root.left = insertRec(root.left, key);
        } else if (key > root.key) {
            root.right = insertRec(root.right, key);
        }
    return root;
}

void delete(int key) {
    root = deleteRec(root, key);
}

Node deleteRec(Node root, int key) {
    if (root == null)
         return root;
    if (key < root.key) {
        root.left = deleteRec(root.left, key);
    } else if (key > root.key) {
        root.right = deleteRec(root.right, key);
    } else {
        if (root.left == null) return root.right;
        else if (root.right == null) return root.left;
       //Delete node with two children using Successor
        root.key = minValue(root.right);
        root.right = deleteRec(root.right, root.key);
    }
    return root;
}

int minValue(Node root) {
    int minv = root.key;
    while (root.left != null) {
        minv = root.left.key;
        root = root.left;
    }
    return minv;
}

int maxValue(Node root) {
    int maxv = root.key;
    while (root.right != null) {
        maxv = root.right.key;
        root = root.right;
    }
    return maxv;
}

Node findPredecessor(Node root, int key) {
    Node predecessor = null;
    while (root != null) {
        if (key > root.key) {
            predecessor = root;
            root = root.right;
        } else {
            root = root.left;
        }
    }
    return predecessor;
}

Node findSuccessor(Node root, int key) {
    Node successor = null;
    while (root != null) {
        if (key < root.key) {
            successor = root;
            root = root.left;
        } else {
            root = root.right;
        }
    }
    return successor;
}

void inorder(Node root) {
    if (root != null) {
        inorder(root.left);
        System.out.print(root.key + " ");
        inorder(root.right);
    }
}

public static void main(String[] args) {
    BST tree = new BST();
    tree.insert(50);
    tree.insert(30);
    tree.insert(70);
    tree.insert(20);
    tree.insert(40);
    tree.insert(60);
```

```
        tree.insert(80);

        System.out.println("Inorder traversal:");
        tree.inorder(tree.root);

        System.out.println("\nMin Value: " + tree.minValue(tree.root));
        System.out.println("Max Value: " + tree.maxValue(tree.root));

        System.out.println("\nDeleting 20");
        tree.delete(20);
        tree.inorder(tree.root);

        Node pred = tree.findPredecessor(tree.root, 50);
        Node succ = tree.findSuccessor(tree.root, 50);

        System.out.println("\nPredecessor of 50: " + (pred != null ? pred.key : "None"));
        System.out.println("Successor of 50: " + (succ != null ? succ.key : "None"));
    }
}
```