

Objective(s):

- To be able to implement binary-search-tree insert(int d) method
- To be able to implement binary tree traversal method
- To be able to implement binary tree search method

Task 1:

Given TreeNode.java and BST.java, complete insert(int d) and preOrder()

```
public class BST {
    TreeNode root;
    public BST() { root = null; }
    // public TreeNode getRoot() {
    //     return root;
    // }
    public void insert(int d) {
        if (root == null) {
            root = new TreeNode(d);
        } else {
            TreeNode cur = root;
            while (cur != null) {
                if (d < cur.data) {
                    if (cur.left != null)
                        cur = cur.left;
                    else {
                        /* your code 1*/
                    }
                } else { //!! (d < p.data)
                    if (cur.right != null)
                        /* your code 2*/;
                    else {
                        cur.right = new TreeNode(d);
                        cur.right.parent = cur;
                        return;
                    }
                }
            }
        } //while
    } //insert by iteration
    public void printPreOrder() {
        printPreOrderRecurse(root);
    }
    private void printPreOrderRecurse(TreeNode node) {
        /* your code 3*/
    }
}
```

```
package code;

public class TreeNode {
    int data;
    TreeNode left, right, parent;

    public TreeNode(int d) {
        data = d;
    }
    @Override
    public String toString() {
        // There are 4 cases no child,
        // left-child-only,
        // right-child-only,
        //and both children
        /* your code 6*/
        return "null<-" + data + "->null";
        // no child
    }
}
```

Note that BST's root cannot be accessed from main, in that case its access modifier should be private and provide getRoot() (commented).

```
public static void demol() {
    println("-insert and preOrder traversal-");
    int[] dat = { 15, 20, 10, 18, 16,
                  12, 8, 25, 19, 30 };

    BST bst = new BST();
    for (int j = 0; j < dat.length; j++)
        bst.insert(dat[j]);

    bst.printPreOrder();
    //15 10 8 12 20 18 16 19 25 30
    System.out.println();
    //demo2(bst);
}
```

Instruction: capture your code for insert(int d) and printPreOrderRecurse (TreeNode node)

```
public void insert(int d) {
    if (root == null) {
        root = new TreeNode(d);
    }
    else {
        TreeNode cur = root;
        while (cur != null) {
            if (d < cur.getData()) {
                if (cur.getLeft() != null) cur = cur.getLeft();
                else { /* your code 1*/
                    cur.left = new TreeNode(d);
                    cur.left.parent = cur;
                    return;
                }
            }
            else { //!(d < p.data)
                if (cur.getRight() != null) { /* your code 2*/
                    cur = cur.getRight();
                }
                else {
                    cur.right = new TreeNode(d);
                    cur.right.parent = cur;
                    return;
                }
            }
        }
    }
}
```

```
private void printPreOrderRecurse(TreeNode node) { /*
    if (node == null) return;
    System.out.print(node.getData() + " ");
    printPreOrderRecurse(node.getLeft());
    printPreOrderRecurse(node.getRight());
}
```

Task 2:

complete `printInOrderRecurse(TreeNode node)` and
`printPostOrderRecurse(TreeNode node)`

confirm your output

Instruction: use the 3 traversal,
 draw bst

```
//uncomment demo2() invocation inside demo1()
static void demo2(BST bst) {
    System.out.println("-more traversal---");
    bst.printInOrder();
    System.out.println();
    // 8 10 12 15 16 18 19 20 25 30

    bst.printPostOrder();
    System.out.println();
    // 8 12 10 16 19 18 30 25 20 15

    // demo3(bst);
}
```

```
public void printInOrder() {
    printInOrderRecurse(root);
}
private void printInOrderRecurse(TreeNode
node) {
    /* your code 4*/
}
public void printPostOrder() {
    printPostOrderRecurse(root);
}
private void
printPostOrderRecurse(TreeNode node) {
    /* your code 5*/
}
```

```
private void printInOrderRecurse(TreeNode node) { /* yo
    if (node == null) return;
    printInOrderRecurse(node.getLeft());
    System.out.print(node.getData() + " ");
    printInOrderRecurse(node.getRight());
}

private void printPostOrderRecurse(TreeNode node) { /*
    if (node == null) return;
    printPostOrderRecurse(node.getLeft());
    printPostOrderRecurse(node.getRight());
    System.out.print(node.getData() + " ");
}
```

Task 3:

In fact, processing `TreeNode` in main is cumbersome (as we preferred encapsulation). However,

```
println("-search recursive---");
println(bst.search(20)); // 18<-20->25
println(bst.search(25)); // null<-25->30
println(bst.search(12)); // null<-12->>null
println(bst.search(1)); // null
println(bst.searchRecurse(10
                        , bst.getRoot()));
//if searchRecurse and getRoot is available

println("-search iterative---");
println(bst.searchIter(20));
println(bst.searchIter(25));
println(bst.searchIter(12));
println(bst.searchIter(1));
```

we'll leave `search(int d)` to return `TreeNode` as is. We'll check the search result in the method.

```
public TreeNode search(int d) {
    TreeNode result = searchRecurse(d, root);
    return result;
}
public TreeNode searchRecurse(int d, TreeNode n) {
    if (n == null) return null;
    if (d == n.data) return n;
    /* your code 7*/
    return searchRecurse(d, n.right);
}
```

```
public TreeNode searchIter(int key) {
    if (root.data == key)
        return root;
    TreeNode current = root;
    while (current != null) {
        if (key < current.data) {
            if (current.left != null)
                current = current.left;
        } else {
            if (current.right != null)
                current = current.right;
        }

        if (current.data == key)
            return current;

        /* your code 8 */
    } //while
    return null;
}
```

Instructions:

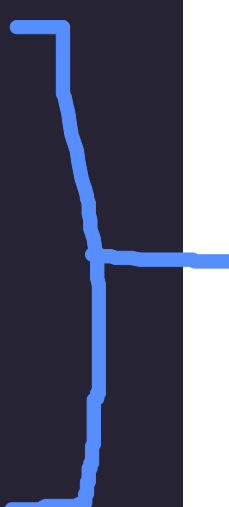
Complete `/* your code 6 */` in `TreeNode.java` so that we can check the search result.

Complete `/* your code 7 */` and `/* your code 8 */`

(The result commented is to confirm your work correctness.)

Capture your `demo3()`'s output

```
-insert and preOrder traversal-
15 10 8 12 20 18 16 19 25 30
-more traversal---
8 10 12 15 16 18 19 20 25 30
8 12 10 16 19 18 30 25 20 15
-search recursive---
18<-20->25
null<-25->30
null<-12->null
null
8<-10->12
-search iterative---
18<-20->25
null<-25->30
null<-12->null
```



Demo 3

Submission: `BST.java` with your student id and name as a comment at the first line of the file.

Due date: TBA