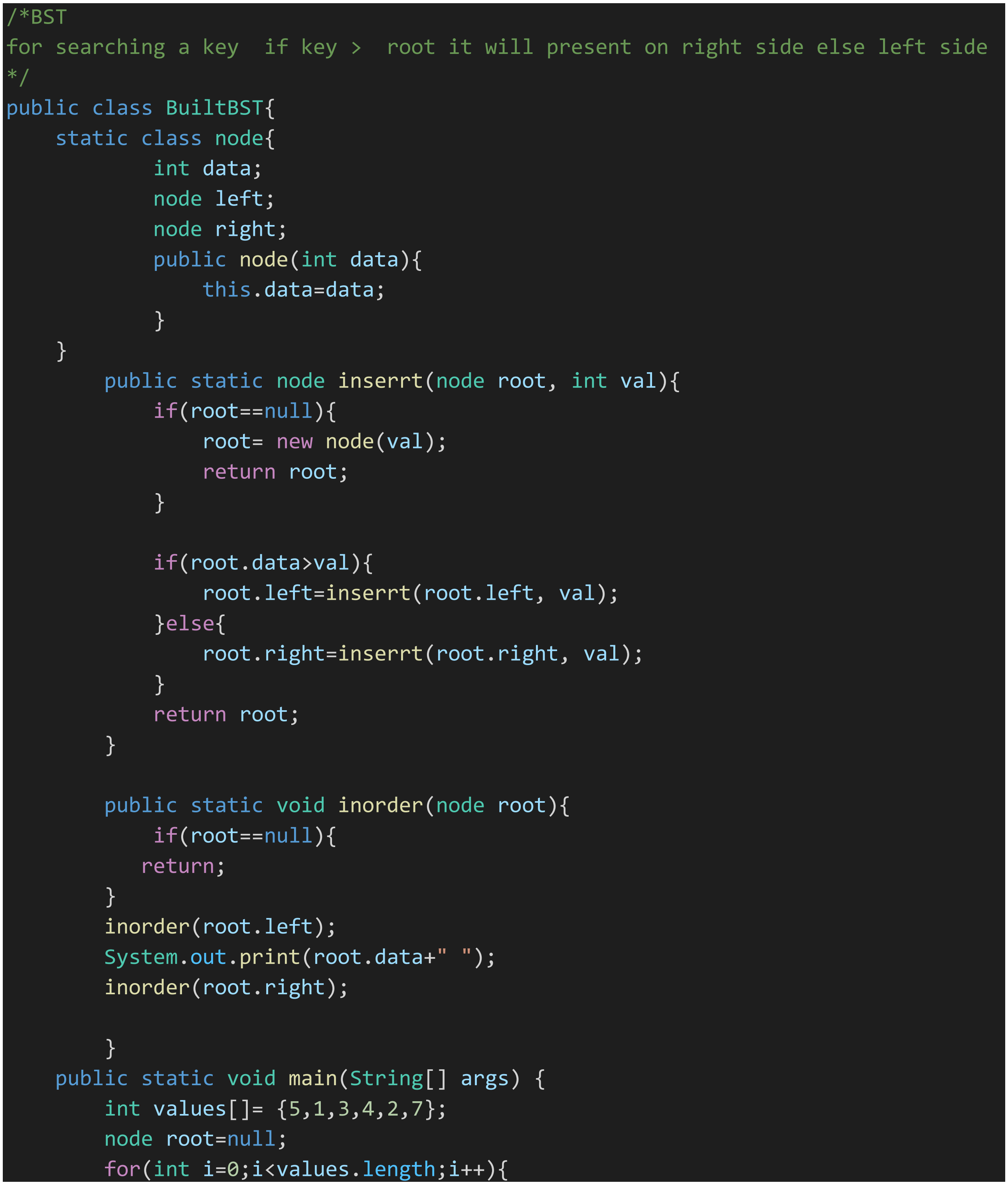
BINARY SEARCH TREES

Code for building and printing BST:



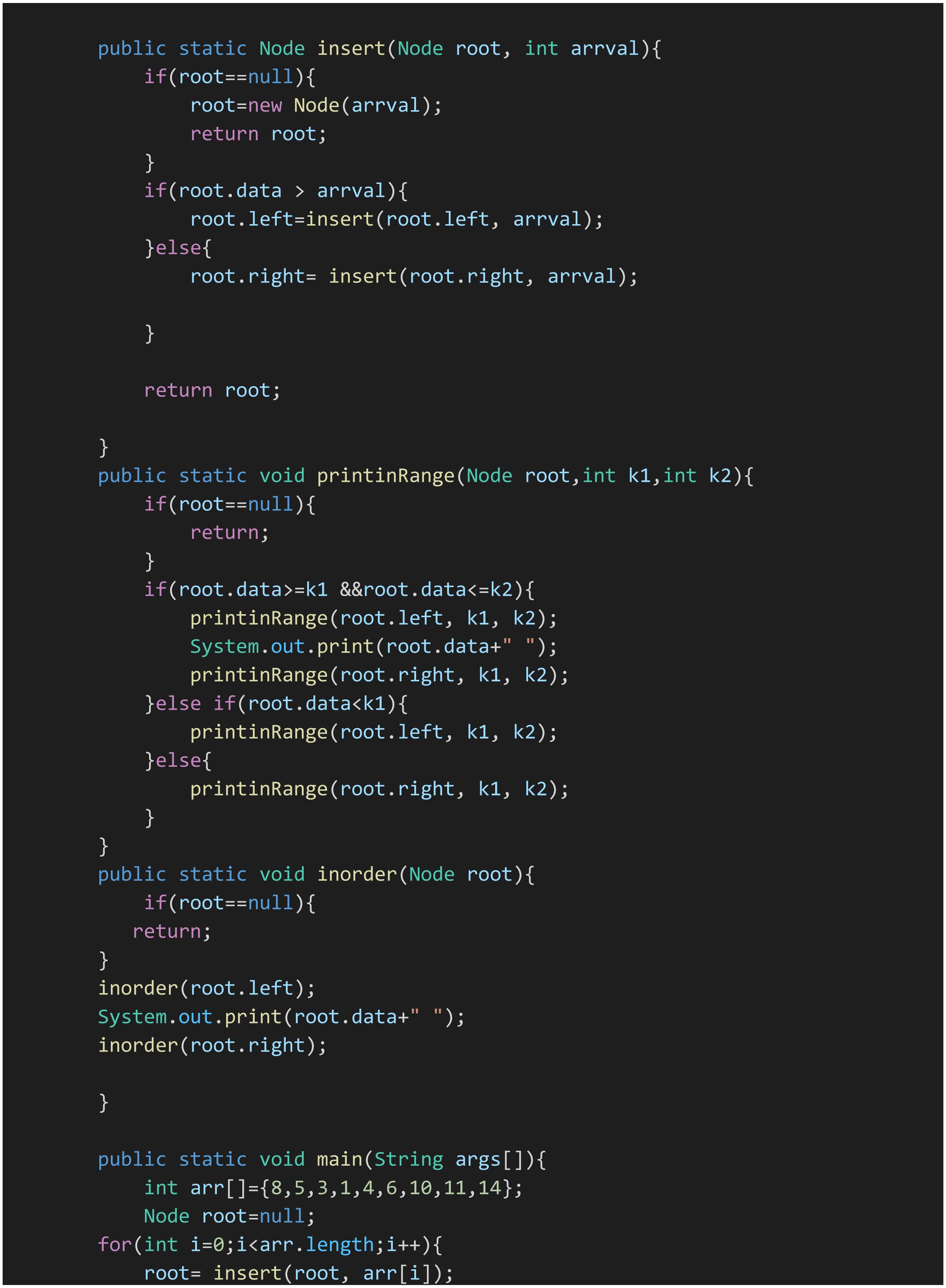
```
/*BST
for searching a key  if key >  root it will present on right side else left side
*/
public class BuiltBST{
    static class node{
            int data;
            node left;
            node right;
            public node(int data){
                this.data=data;
            }
    }
        public static node inserrt(node root, int val){
            if(root==null){
                root= new node(val);
                return root;
            }

            if(root.data>val){
                root.left=inserrt(root.left, val);
            }else{
                root.right=inserrt(root.right, val);
            }
            return root;
        }

        public static void inorder(node root){
            if(root==null){
           return;
        }
        inorder(root.left);
        System.out.print(root.data+" ");
        inorder(root.right);

        }
    public static void main(String[] args) {
        int values[]= {5,1,3,4,2,7};
        node root=null;
        for(int i=0;i<values.length;i++){
            root=inserrt(root, values[i]);
        }
        inorder(root);

    }
}
Output:
1,2,3,4,5,7
```

Code for searching a key in BST:

```
public class BSTsearch {
    static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data= data;
            this.left=null;
            this.right=null;
        }
    }

    public static Node insert(Node root, int arrval){
        if(root==null){
            root=new Node(arrval);
            return root;
        }
        if(root.data > arrval){
            root.left=insert(root.left, arrval);
        }else{
            root.right= insert(root.right, arrval);

        }

        return root;

    }
    public static boolean isfound(Node root,int key){
        if(root==null){
            return false;
        }
        if(root.data==key){
            return true;
        }
        if(root.data > key){
            return isfound(root.left, key);
        }else{
           return  isfound(root.right, key);
        }

    }
    public static void main(String args[]){
        int arr[]={5,1,3,4,2,7};

        Node root=null;
    for(int i=0;i<arr.length;i++){
        root= insert(root, arr[i]);
    }

       if(isfound(root,6)){
        System.out.println("found");
       }
       else{
        System.out.println( "not found");
       }
    }

}
Output:
Not found
```

Print in range:

```
public class printinRange {

        static class Node{
            int data;
            Node left;
            Node right;
            Node(int data){
                this.data= data;
                this.left=null;
                this.right=null;
            }
        }
```



```java
public static Node insert(Node root, int arrval){
    if(root==null){
        root=new Node(arrval);
        return root;
    }
    if(root.data > arrval){
        root.left=insert(root.left, arrval);
    }else{
        root.right= insert(root.right, arrval);

    }

    return root;

}
public static void printinRange(Node root,int k1,int k2){
    if(root==null){
        return;
    }
    if(root.data>=k1 &&root.data<=k2){
        printinRange(root.left, k1, k2);
        System.out.print(root.data+" ");
        printinRange(root.right, k1, k2);
    }else if(root.data<k1){
        printinRange(root.left, k1, k2);
    }else{
        printinRange(root.right, k1, k2);
    }
}
public static void inorder(Node root){
    if(root==null){
   return;
}
inorder(root.left);
System.out.print(root.data+" ");
inorder(root.right);

}

public static void main(String args[]){
    int arr[]={8,5,3,1,4,6,10,11,14};
    Node root=null;
for(int i=0;i<arr.length;i++){
    root= insert(root, arr[i]);
```

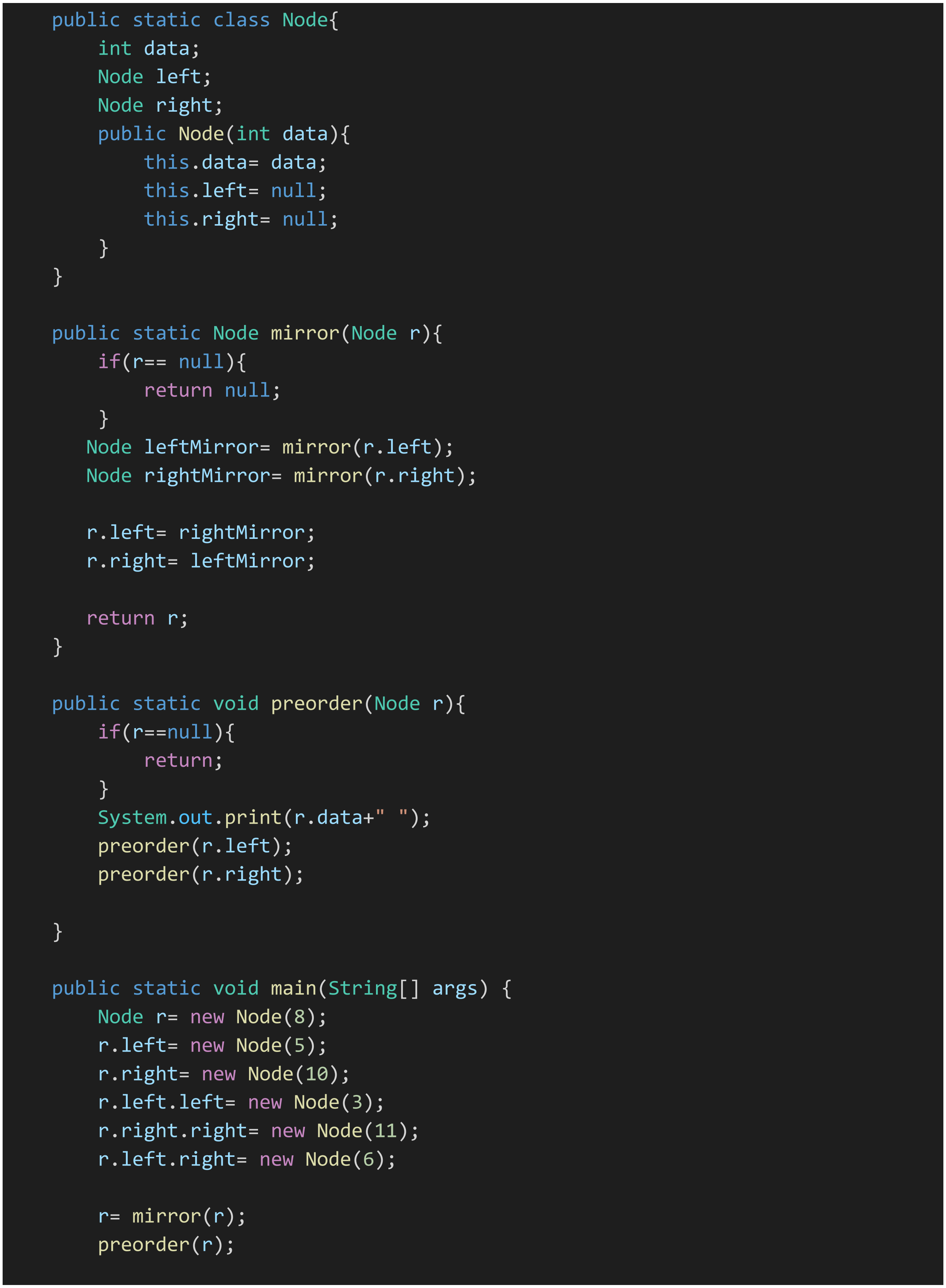
```
        }
        // inorder(root);
        printinRange(root, 5, 12);


}}output:
5 6 8 10 11
```

VALID BINARY SEARCH TREE:

```
public class validBST {

        static class Node{
            int data;
            Node left;
            Node right;
            Node(int data){
                this.data= data;
                this.left=null;
                this.right=null;
            }
        }

        public static Node insert(Node root, int arrval){
            if(root==null){
                root=new Node(arrval);
                return root;
            }
            if(root.data > arrval){
                root.left=insert(root.left, arrval);
            }else{
                root.right= insert(root.right, arrval);

            }

            return root;

        }

        public static void inorder(Node root){
            if(root==null){
           return;
        }
        inorder(root.left);
        System.out.print(root.data+" ");
        inorder(root.right);

        }

    public static boolean isvalid(Node root, Node max, Node min)  {
        if(root==null){
            return true;
        }

        else if(min!=null&& root.data<=min.data){
            return false;
        }
        else if(max != null&&root.data>=max.data){
            return false;
        }

        return isvalid(root.right,root , max) && isvalid(root.left,min ,root);
    }
        public static void main(String args[]){
            int arr[]={1,1,1};
            Node root=null;
        for(int i=0;i<arr.length;i++){
            root= insert(root, arr[i]);
        }
       if(isvalid(root, null, null)){
        System.out.println("valid");
       }
       else{
        System.out.println("not valid");
       }




}
}
Output:
Not valid
```

Mirror of a BST:

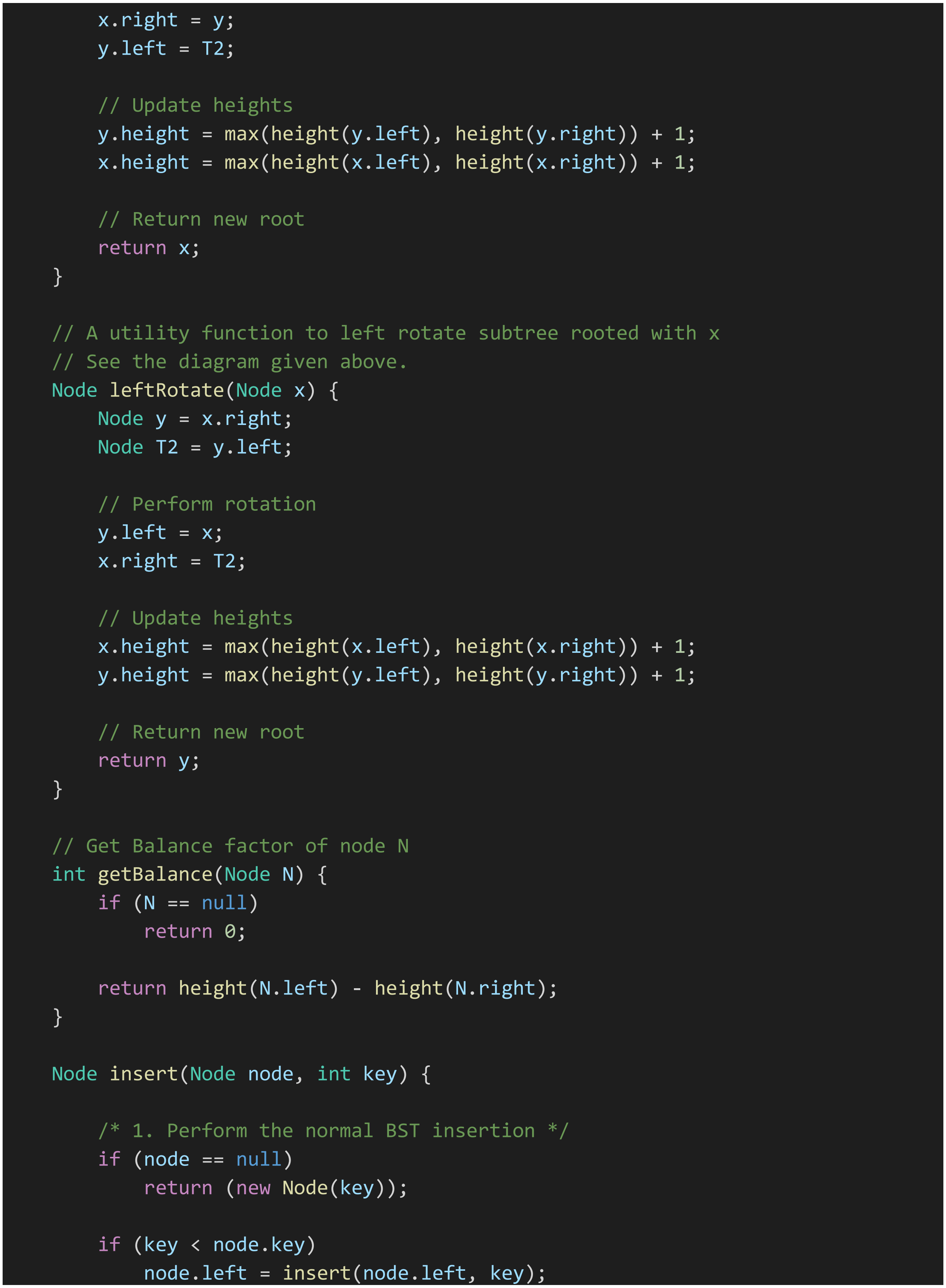
```
public class MIrrorOfBST {
```



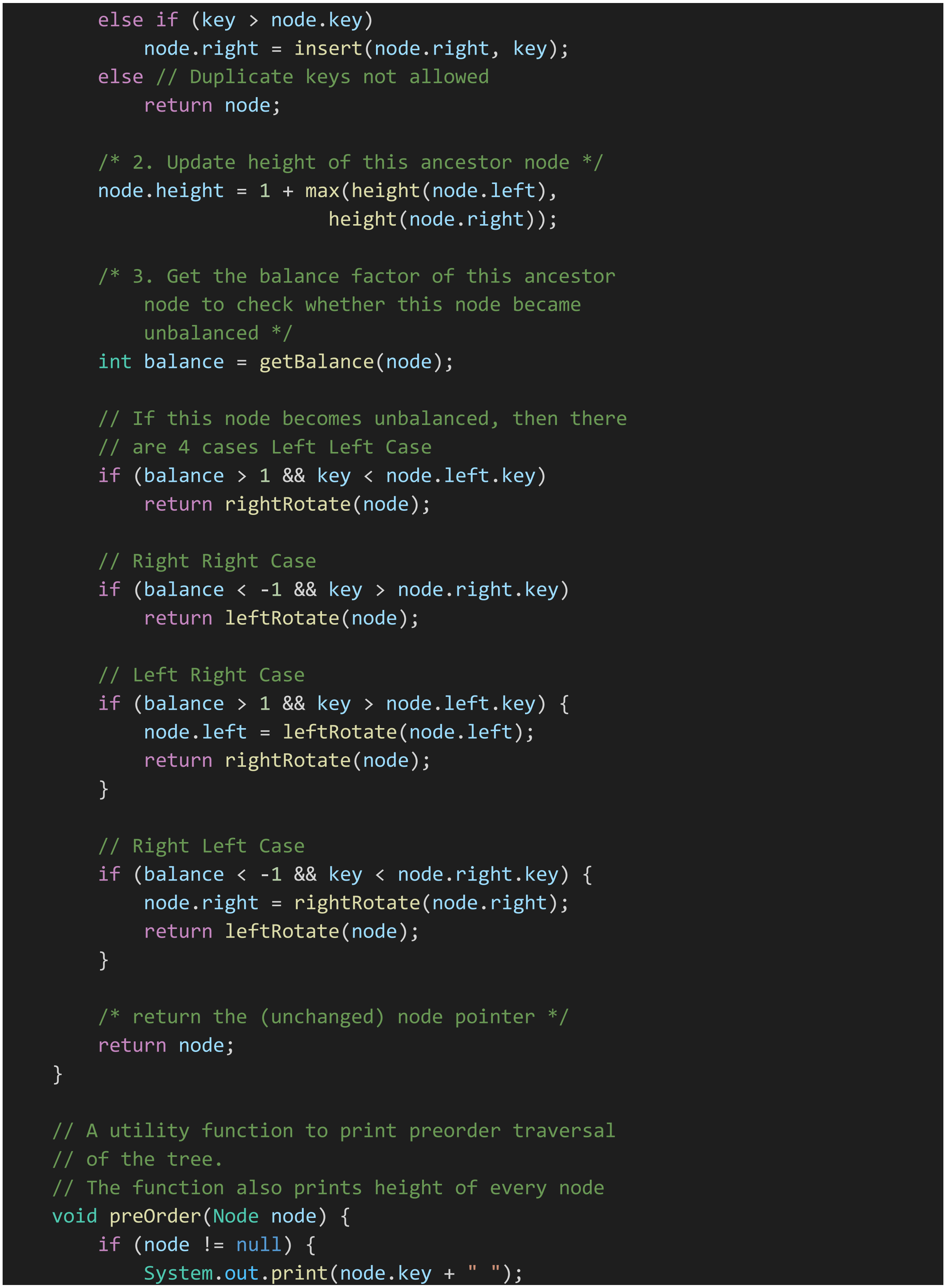
```
public static class Node{
    int data;
    Node left;
    Node right;
    public Node(int data){
        this.data= data;
        this.left= null;
        this.right= null;
    }
}

public static Node mirror(Node r){
    if(r== null){
        return null;
    }
   Node leftMirror= mirror(r.left);
   Node rightMirror= mirror(r.right);

   r.left= rightMirror;
   r.right= leftMirror;

   return r;
}

public static void preorder(Node r){
    if(r==null){
        return;
    }
    System.out.print(r.data+" ");
    preorder(r.left);
    preorder(r.right);

}

public static void main(String[] args) {
    Node r= new Node(8);
    r.left= new Node(5);
    r.right= new Node(10);
    r.left.left= new Node(3);
    r.right.right= new Node(11);
    r.left.right= new Node(6);

    r= mirror(r);
    preorder(r);
    }
}
Output:
8 10 11 5 6 3
```

Code for AVL TREES:

```
// Java program for insertion in AVL Tree
class Node {
    int key, height;
    Node left, right;

    Node(int d) {
        key = d;
        height = 1;
    }
}

class AVLTree {

    Node root;

    // A utility function to get the height of the tree
    int height(Node N) {
        if (N == null)
            return 0;

        return N.height;
    }

    // A utility function to get maximum of two integers
    int max(int a, int b) {
        return (a > b) ? a : b;
    }

    // A utility function to right rotate subtree rooted with y
    // See the diagram given above.
    Node rightRotate(Node y) {
        Node x = y.left;
        Node T2 = x.right;

        // Perform rotation
```



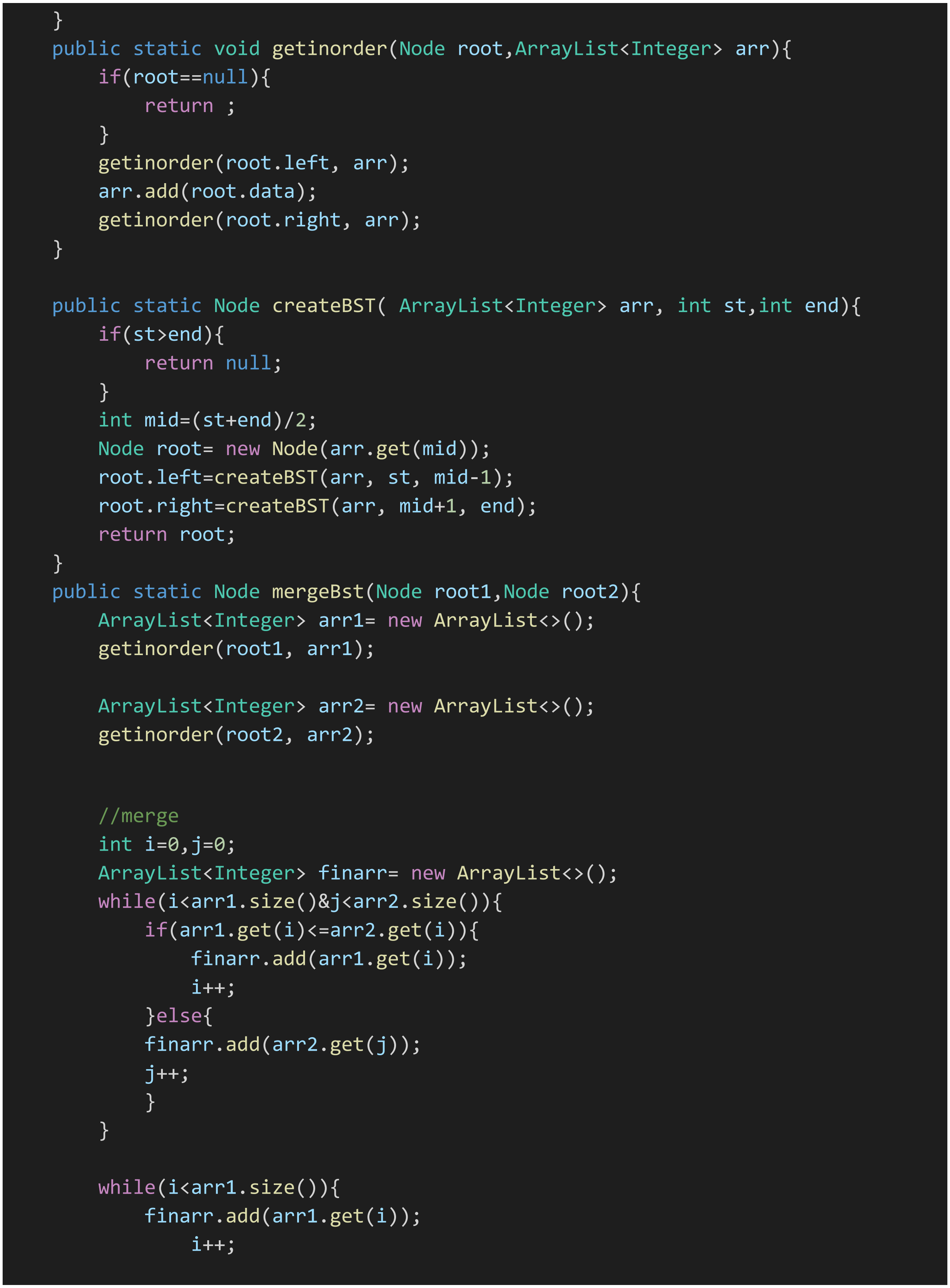
```
        x.right = y;
        y.left = T2;

        // Update heights
        y.height = max(height(y.left), height(y.right)) + 1;
        x.height = max(height(x.left), height(x.right)) + 1;

        // Return new root
        return x;
    }

    // A utility function to left rotate subtree rooted with x
    // See the diagram given above.
    Node leftRotate(Node x) {
        Node y = x.right;
        Node T2 = y.left;

        // Perform rotation
        y.left = x;
        x.right = T2;

        // Update heights
        x.height = max(height(x.left), height(x.right)) + 1;
        y.height = max(height(y.left), height(y.right)) + 1;

        // Return new root
        return y;
    }

    // Get Balance factor of node N
    int getBalance(Node N) {
        if (N == null)
            return 0;

        return height(N.left) - height(N.right);
    }

    Node insert(Node node, int key) {

        /* 1. Perform the normal BST insertion */
        if (node == null)
            return (new Node(key));

        if (key < node.key)
            node.left = insert(node.left, key);
```



```
        else if (key > node.key)
            node.right = insert(node.right, key);
        else // Duplicate keys not allowed
            return node;

        /* 2. Update height of this ancestor node */
        node.height = 1 + max(height(node.left),
                            height(node.right));

        /* 3. Get the balance factor of this ancestor
            node to check whether this node became
            unbalanced */
        int balance = getBalance(node);

        // If this node becomes unbalanced, then there
        // are 4 cases Left Left Case
        if (balance > 1 && key < node.left.key)
            return rightRotate(node);

        // Right Right Case
        if (balance < -1 && key > node.right.key)
            return leftRotate(node);

        // Left Right Case
        if (balance > 1 && key > node.left.key) {
            node.left = leftRotate(node.left);
            return rightRotate(node);
        }

        // Right Left Case
        if (balance < -1 && key < node.right.key) {
            node.right = rightRotate(node.right);
            return leftRotate(node);
        }

        /* return the (unchanged) node pointer */
        return node;
    }

    // A utility function to print preorder traversal
    // of the tree.
    // The function also prints height of every node
    void preOrder(Node node) {
        if (node != null) {
            System.out.print(node.key + " ");
            preOrder(node.left);
            preOrder(node.right);
        }
    }

    public static void main(String[] args) {
        AVLTree tree = new AVLTree();

        /* Constructing tree given in the above figure */
        tree.root = tree.insert(tree.root, 10);
        tree.root = tree.insert(tree.root, 20);
        tree.root = tree.insert(tree.root, 30);
        tree.root = tree.insert(tree.root, 40);
        tree.root = tree.insert(tree.root, 50);
        tree.root = tree.insert(tree.root, 25);

        /* The constructed AVL Tree would be
            30
            / \
        20 40
        / \  \
        10 25 50
        */

        tree.preOrder(tree.root);
    }
}
Output:
30 20 10 25 40 50
```

Code for Merging 2 bst:

```
import java.util.ArrayList;

public class merge2BST {

    static class Node{
        int data;
        Node left;
        Node right;
        public Node(int data){
            this.data= data;
            this.left= null;
            this.right= null;
        }
```



```
    }
    public static void getinorder(Node root,ArrayList<Integer> arr){
        if(root==null){
            return ;
        }
        getinorder(root.left, arr);
        arr.add(root.data);
        getinorder(root.right, arr);
    }

    public static Node createBST( ArrayList<Integer> arr, int st,int end){
        if(st>end){
            return null;
        }
        int mid=(st+end)/2;
        Node root= new Node(arr.get(mid));
        root.left=createBST(arr, st, mid-1);
        root.right=createBST(arr, mid+1, end);
        return root;
    }
    public static Node mergeBst(Node root1,Node root2){
        ArrayList<Integer> arr1= new ArrayList<>();
        getinorder(root1, arr1);

        ArrayList<Integer> arr2= new ArrayList<>();
        getinorder(root2, arr2);


        //merge
        int i=0,j=0;
        ArrayList<Integer> finarr= new ArrayList<>();
        while(i<arr1.size()&j<arr2.size()){
            if(arr1.get(i)<=arr2.get(i)){
                finarr.add(arr1.get(i));
                i++;
            }else{
            finarr.add(arr2.get(j));
            j++;
            }
        }

        while(i<arr1.size()){
            finarr.add(arr1.get(i));
                i++;
        }
        while(j<arr2.size()){
            finarr.add(arr2.get(j));
            j++;
        }

        return createBST(finarr,0,finarr.size()-1);
    }

    public static void preorder(Node root){
        if(root==null){
            return;
        }
        System.out.print(root.data+" ");
        preorder(root.left);
        preorder(root.right);
    }

    public static void main(String args[]){
        Node root1= new Node(2);
        root1.left= new Node(1);
        root1.right= new Node(4);


        Node root2= new Node(9);
        root2.left=new Node(3);
        root2.right= new Node(12);

        Node root= mergeBst(root1, root2);
        preorder(root);
    }

}
Output:
4 1 2 9 3 12
```