```
import java.util.LinkedList;

public class BinarySearchTreeDemo
{
    public static void main(String[] args)
    {
        TreeNode root = new TreeNode(50);
        TreeNode newNode;
        String originalOrder = "";

        for (int i = 0; i < 10; i++)
        {
            int nextValue = (int) (Math.random() * 100) + 1;
            newNode = new TreeNode(nextValue);
            insertNode(root, newNode);
            originalOrder += " " + nextValue;
        }

        System.out.println("\nnodes in original order: " +
                           root.getValue() + " " + originalOrder + "\n");

        System.out.println("\nnodes inorder:");
        printNodesInOrder(root);
        System.out.println("\nnodes in level order:");
        printNodesLevelOrder(root);
        System.out.println("\nnodes in preorder:");
        printNodesPreOrder(root);
        System.out.println("\nnodes in postorder:");
        printNodesPostOrder(root);

        int randNum = (int) (Math.random() * 100) + 1;

        if (findNode(root, randNum))
            System.out.println("\n" + randNum + " was found");
        else
            System.out.println("\n" + randNum + " was not found");

        TreeNode temp = root.getRight().getLeft();

        if (isLeaf(temp))
            System.out.println(temp.getValue() + " is a leaf");
        else
            System.out.println(temp.getValue() + " is not a leaf");

        System.out.println("\ntree depth = " + treeDepth(root));
        System.out.println("\ntree sum = " + treeSum(root));
        System.out.println("\nint. path length = " +
            internalPathLength(root,1));
        System.out.println("\nnumber of leafs = " + countLeafs(root));
        System.out.println("\nsmallest integer = " + findMin(root));
        System.out.println("\nnumber of nodes = " + nodeCount(root));

         // testing deleteNode method
        int testValue = 1;
        insertNode(root, new TreeNode(testValue));
        System.out.println("\nnodes in order with test value " + testValue + ":");
        printNodesInOrder(root);
        deleteNode(testValue, root);
        System.out.println("\nnodes in order with test value deleted:");
        printNodesInOrder(root);
    }
```

```
public static void insertNode(TreeNode current, TreeNode newNode)
{
    Comparable newValue = (Comparable) newNode.getValue();
    Comparable currValue = (Comparable) current.getValue();

    if (newValue.compareTo(currValue) < 0)
    {
        if (current.getLeft() == null)
            current.setLeft(newNode);
        else
            insertNode(current.getLeft(), newNode);
    }
    else
    {
        if (current.getRight() == null)
            current.setRight(newNode);
        else
            insertNode(current.getRight(), newNode);
    }
}

public static boolean findNode(TreeNode current, Comparable key)
{
    if (current == null)
        return false;
    else
    {
        Comparable currValue = (Comparable) current.getValue();
        if (key.compareTo(currValue) == 0)
            return true;
        else if (key.compareTo(currValue) < 0)
            return findNode(current.getLeft(), key);
        else
            return findNode(current.getRight(), key);
    }
}

public static TreeNode deleteNode(Comparable x, TreeNode t)
{
    if( t == null )
    {
        return t;
    }
    else if(x.compareTo(t.getValue()) < 0)
    {
        t.setLeft(deleteNode( x, t.getLeft()));
    }
    else if(x.compareTo(t.getValue()) > 0)
    {
        t.setRight(deleteNode(x, t.getRight()));
    }
    else if(t.getLeft() != null && t.getRight() != null)
    {
        t.setValue(findMin(t.getRight()));
        t.setRight(deleteNode( (Comparable)t.getValue(), t.getRight()));
    }
    else
    {
        if (t.getLeft() != null)
        {
            t = t.getLeft();
        }
```

```
        else
        {
            t = t.getRight();
        }
    }

    return t;
}

public static int treeDepth(TreeNode root)
{
    /*
    int depth = 0;
    if (root == null)
        return 0;
    if (root.getRight() == null && root.getLeft() == null)
        return depth;
    if (treeDepth(root.getRight()) > treeDepth(root.getLeft()))
        return depth + treeDepth(root.getRight()) + 1;
    return depth + treeDepth(root.getLeft()) + 1;
    */ // OR
    if (root == null)
        return -1;
    else
        return 1 + Math.max(treeDepth(root.getLeft()), treeDepth(root.getRight()));
}

public static int nodeCount(TreeNode root)
{
    if (root == null)
        return 0;
    return 1 + nodeCount(root.getLeft()) +
        nodeCount(root.getRight());
}

public static int treeSum(TreeNode root)
{
    if (root == null)
        return 0;
    return ((Integer) root.getValue()).intValue() +
        treeSum(root.getLeft()) + treeSum(root.getRight());
}

public static int countLeafs(TreeNode root)
{
    if (root == null)
        return 0;
    else if (root.getLeft() == null && root.getRight() == null)
        return 1;
    return countLeafs(root.getLeft()) + countLeafs(root.getRight());
}

public static int findMin(TreeNode root)
{
    if (root.getLeft() == null)
        return ((Integer) root.getValue()).intValue();
    return findMin(root.getLeft());
}

public static void printNodesInOrder(TreeNode root)
{
    if (root != null)
```

```
    {
        printNodesInOrder(root.getLeft());
        System.out.println(root.getValue());
        printNodesInOrder(root.getRight());
    }
}

public static void printNodesPostOrder(TreeNode root)
{
    if (root != null)
    {
        printNodesPostOrder(root.getLeft());
        printNodesPostOrder(root.getRight());
        System.out.println(root.getValue());
    }
}

public static void printNodesPreOrder(TreeNode root)
{
    if (root != null)
    {
        System.out.println(root.getValue());
        printNodesPreOrder(root.getLeft());
        printNodesPreOrder(root.getRight());
    }
}

public static void printNodesLevelOrder(TreeNode root)
{
    Queue<TreeNode> temp = new LinkedList<TreeNode>();
    if (root != null)
    {
        temp.add(root);
        while (!temp.isEmpty())
        {
            root = temp.remove();
            System.out.println(root.getValue());
            if (root.getLeft() != null)
                temp.add(root.getLeft());
            if (root.getRight() != null)
                temp.add(root.getRight());
        }
    }
}

public static int internalPathLength(TreeNode root, int pathsSoFar)
{
    if (root == null)
        return 0;
    else
        return pathsSoFar + internalPathLength(root.getLeft(),
            pathsSoFar + 1)
            + internalPathLength(root.getRight(), pathsSoFar+1);
}

public static boolean isLeaf(TreeNode root)
{
    if (root != null && root.getRight() == null &&
        root.getLeft() == null)
        return true;
    return false;
}
```

```
}

class TreeNode
{
  private Object value;
  private TreeNode left;
  private TreeNode right;

  public TreeNode(Object initValue)
  {
    value = initValue;
    left = null;
    right = null;
  }

  public TreeNode(Object initValue, TreeNode initLeft, TreeNode initRight)
  {
    value = initValue;
    left = initLeft;
    right = initRight;
  }

  public Object getValue()
  {
    return value;
  }

  public TreeNode getLeft()
  {
    return left;
  }

  public TreeNode getRight()
  {
    return right;
  }

  public void setValue(Object theNewValue)
  {
    value = theNewValue;
  }

  public void setLeft(TreeNode theNewLeft)
  {
    left = theNewLeft;
  }

  public void setRight(TreeNode theNewRight)
  {
    right = theNewRight;
  }
}
```