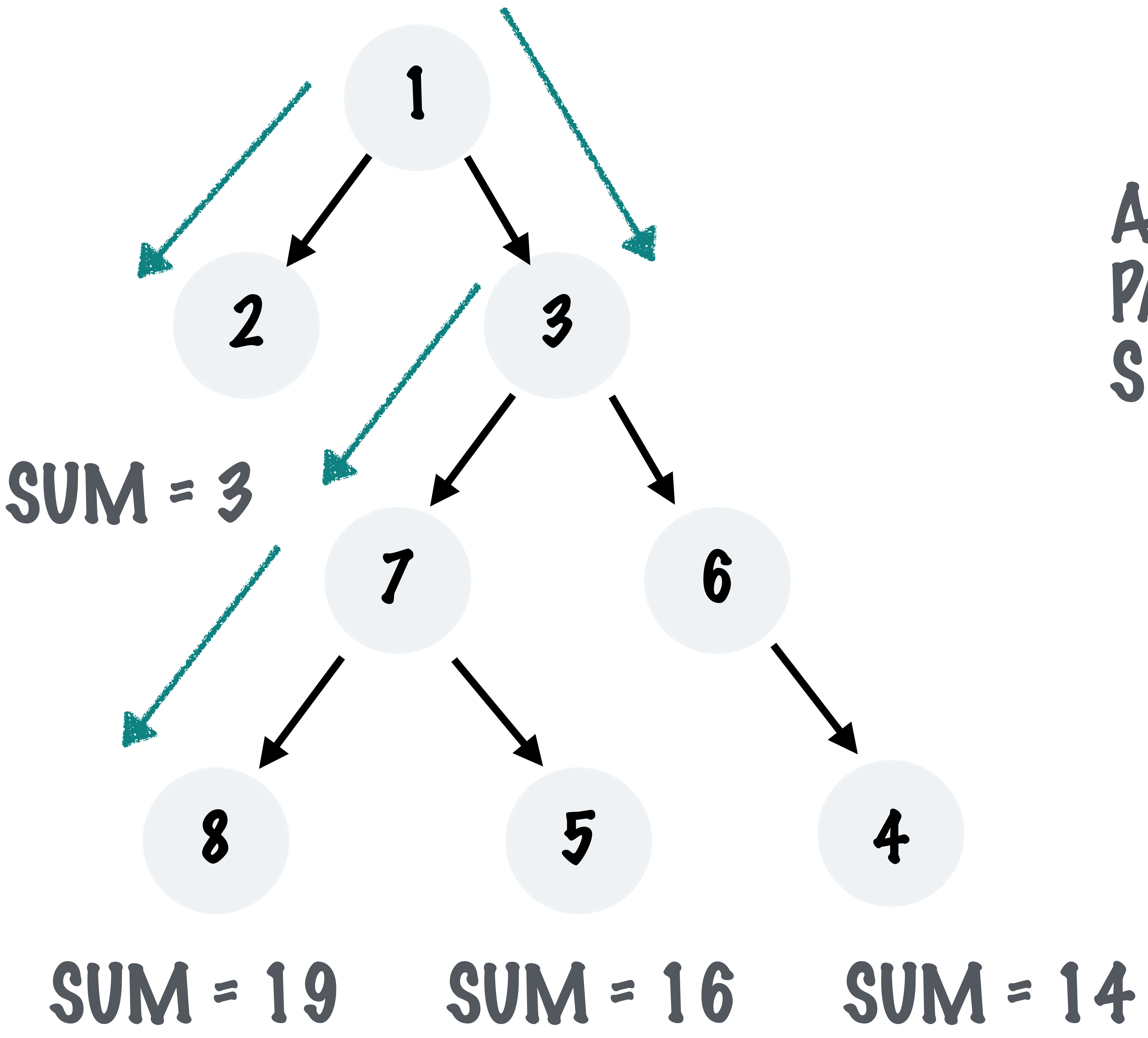
CHECK IF A PATH FROM ROOT TO LEAF NODE SUMS UP TO A CERTAIN VALUE

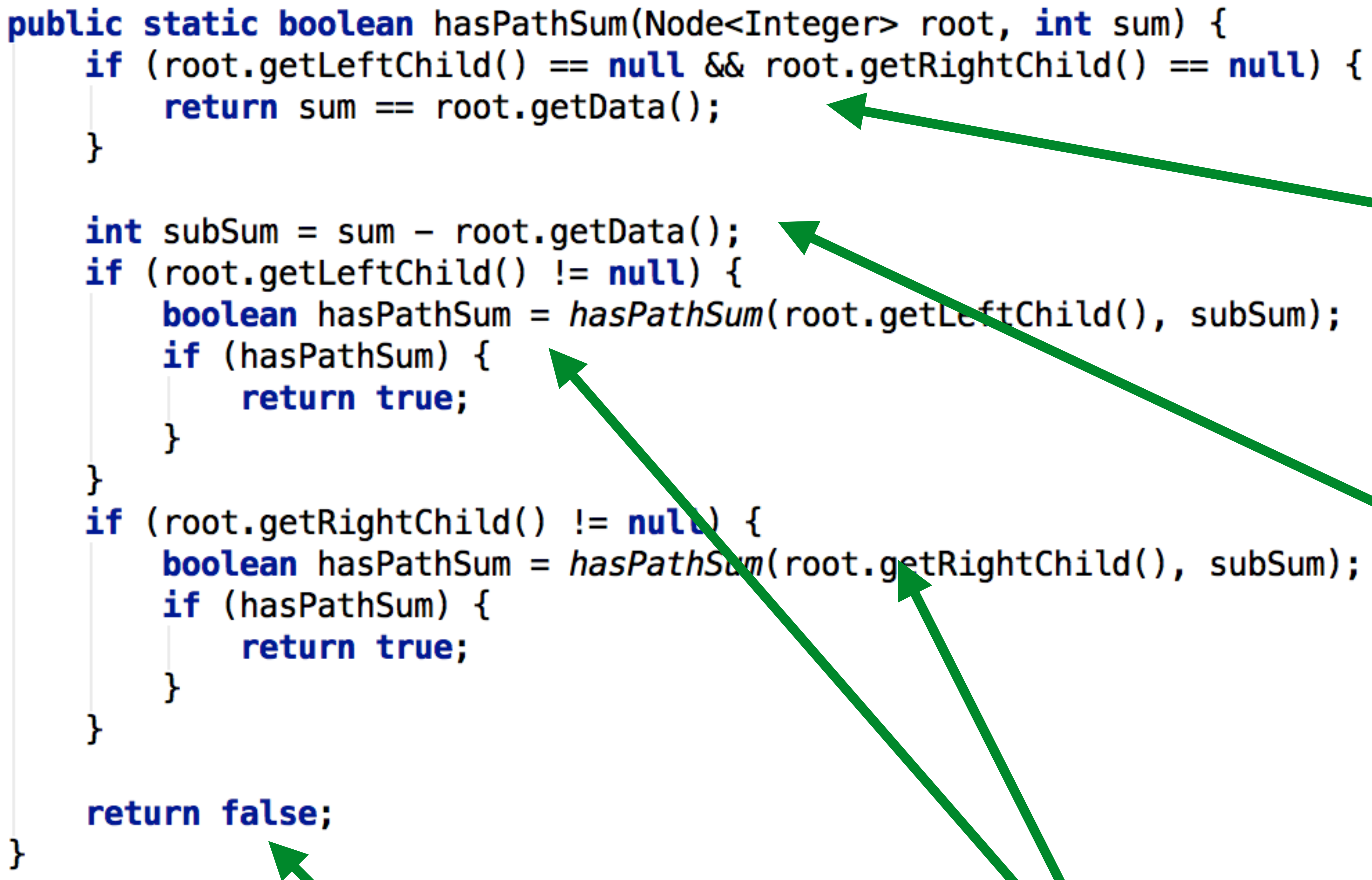
CHECK IF A PATH FROM ROOT TO LEAF NODE SUMS UP TO A CERTAIN VALUE



AT EVERY LEAF NODE CHECK IF THE PATH TO IT SUMS TO THE VALUE SPECIFIED

SUBTRACT THE CURRENT NODE'S VALUE FROM THE SUM WHEN RECURSING LEFT AND RIGHT TOWARDS THE LEAF NODE

HAS PATH SUM?



```java
public static boolean hasPathSum(Node<Integer> root, int sum) {
    if (root.getLeftChild() == null && root.getRightChild() == null) {
        return sum == root.getData();
    }

    int subSum = sum - root.getData();
    if (root.getLeftChild() != null) {
        boolean hasPathSum = hasPathSum(root.getLeftChild(), subSum);
        if (hasPathSum) {
            return true;
        }
    }
    if (root.getRightChild() != null) {
        boolean hasPathSum = hasPathSum(root.getRightChild(), subSum);
        if (hasPathSum) {
            return true;
        }
    }

    return false;
}
```

PASS IN THE CURRENT RUNNING SUM

IN THE CASE OF A LEAF NODE, CHECK IF THE SUM IS EXACTLY EQUAL TO THE VALUE OF THE NODE

FOR INTERNAL, NON-LEAF NODES SUBTRACT THE CURRENT NODE VALUE FROM THE SUM

RETURN FALSE IF THE SUM HAS NOT BEEN FOUND ALONG ANY OF THE SUB TREES

RECURSE LEFT AND RIGHT TO SEE IF THE SUB SUM IS SATISFIED IN ANY OF THE PATHS IN THE RIGHT AND LEFT SUBTREES

PRINT ALL PATHS FROM THE ROOT TO THE LEAF NODES

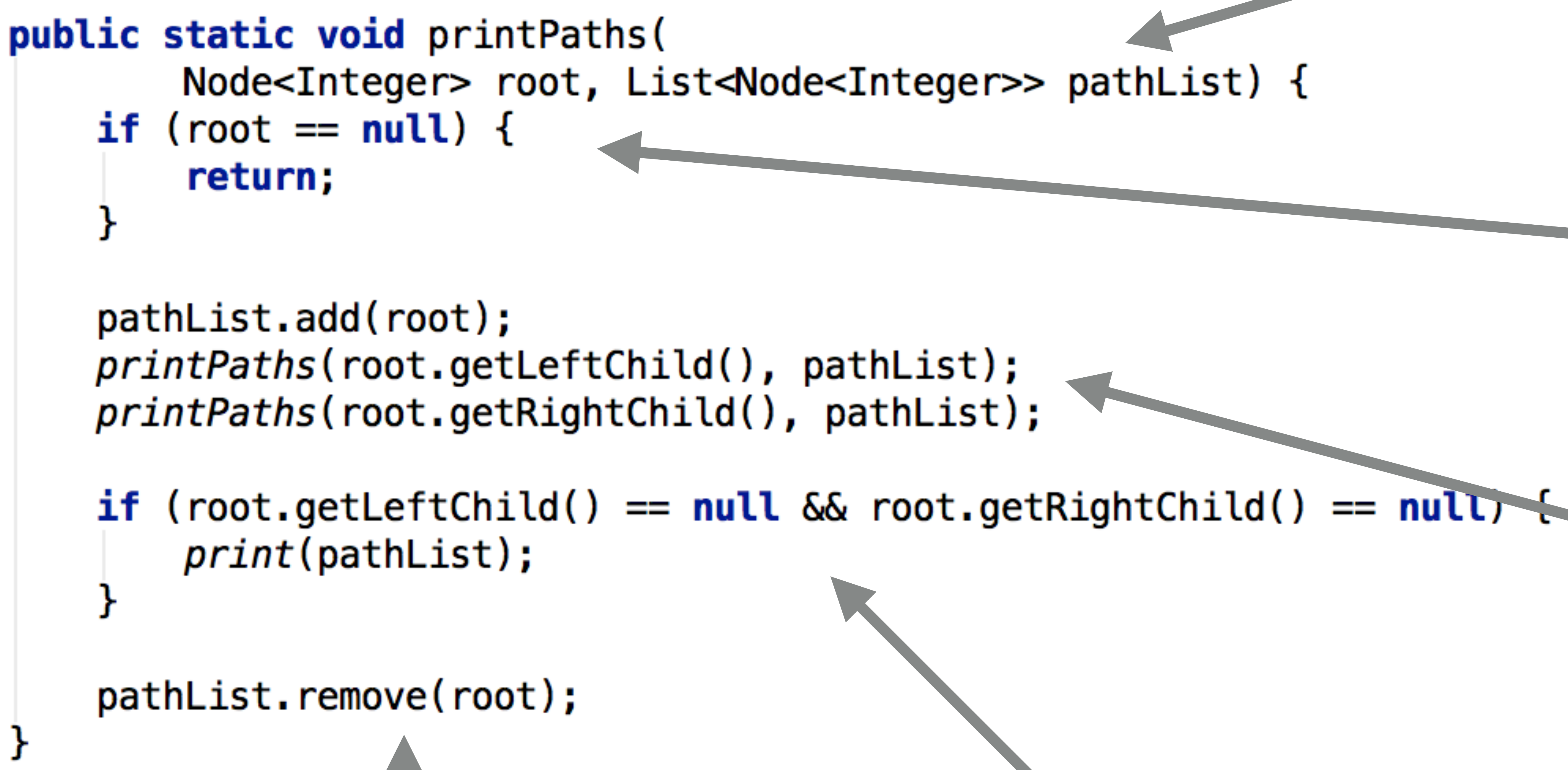
PRINT ALL PATHS FROM THE ROOT TO THE LEAF NODES

KEEP TRACK OF THE CURRENT PATH FOLLOWED TO REACH THE LEAF NODE

AT A LEAF NODE - PRINT THE CURRENT PATH

FOR INTERNAL NODES ADD THE NODE TO THE PATH AND RECURSE TO THE LEFT AND RIGHT CHILDREN

PRINT PATHS



```java
public static void printPaths(
        Node<Integer> root, List<Node<Integer>> pathList) {
    if (root == null) {
        return;
    }

    pathList.add(root);
    printPaths(root.getLeftChild(), pathList);
    printPaths(root.getRightChild(), pathList);

    if (root.getLeftChild() == null && root.getRightChild() == null) {
        print(pathList);
    }

    pathList.remove(root);
}
```

A LIST KEEPING TRACK OF THE CURRENT PATH TO THIS NODE

A NULL ROOT, NOTHING TO DO

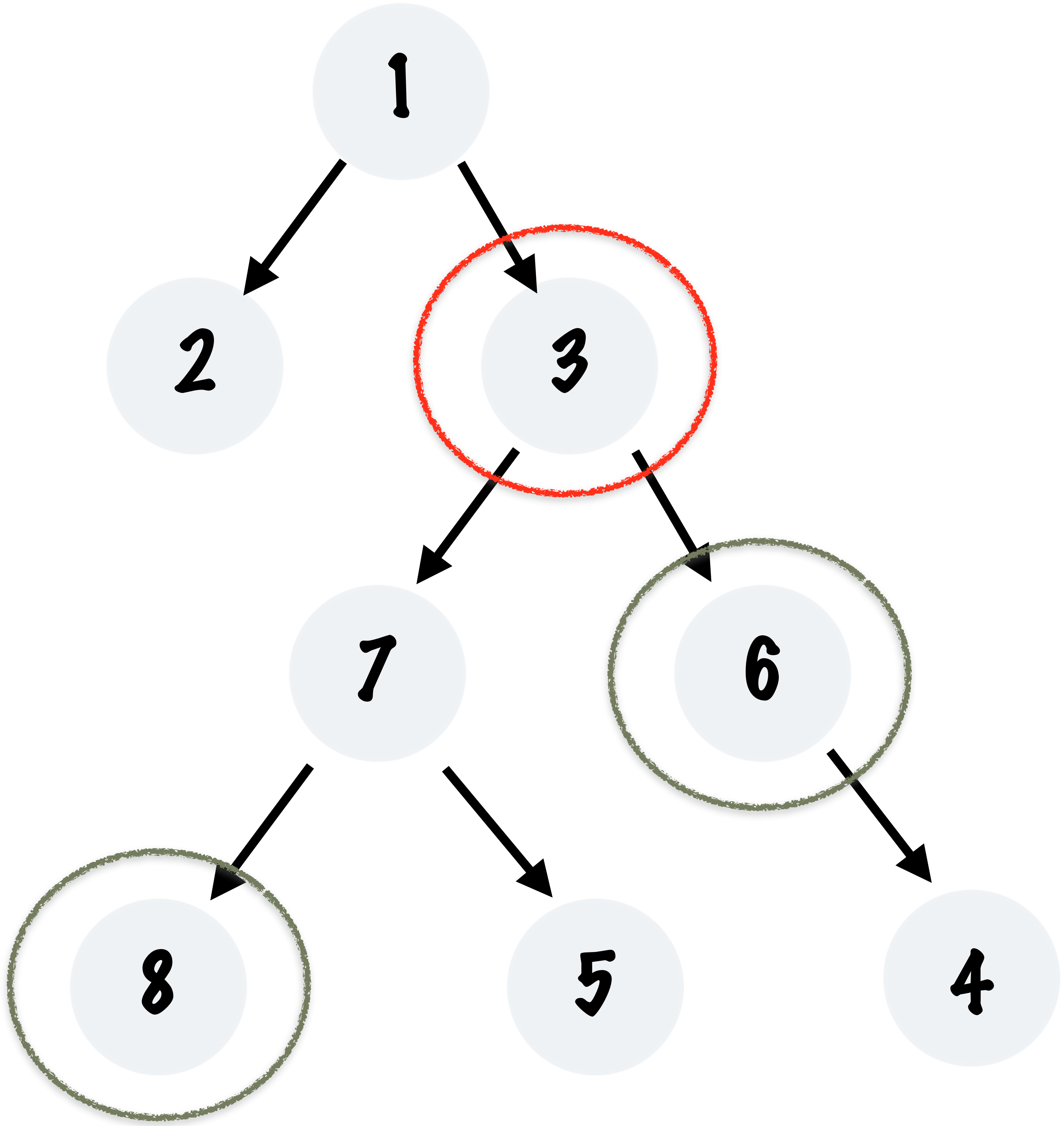
ADD THE CURRENT NODE TO THE PATH AND RECURSE TO THE LEFT AND RIGHT CHILD

IF THIS IS A LEAF NODE, PRINT THE CURRENT PATH, WHICH HAS ALL THE NODES LEADING TO THIS LEAF NODE

REMOVE THE CURRENT NODE FROM THE PATH LIST AS ALL PATHS FROM THIS NODE HAVE BEEN PROCESSED AND PRINTED

FIND THE LEAST COMMON ANCESTOR FOR 2 NODES

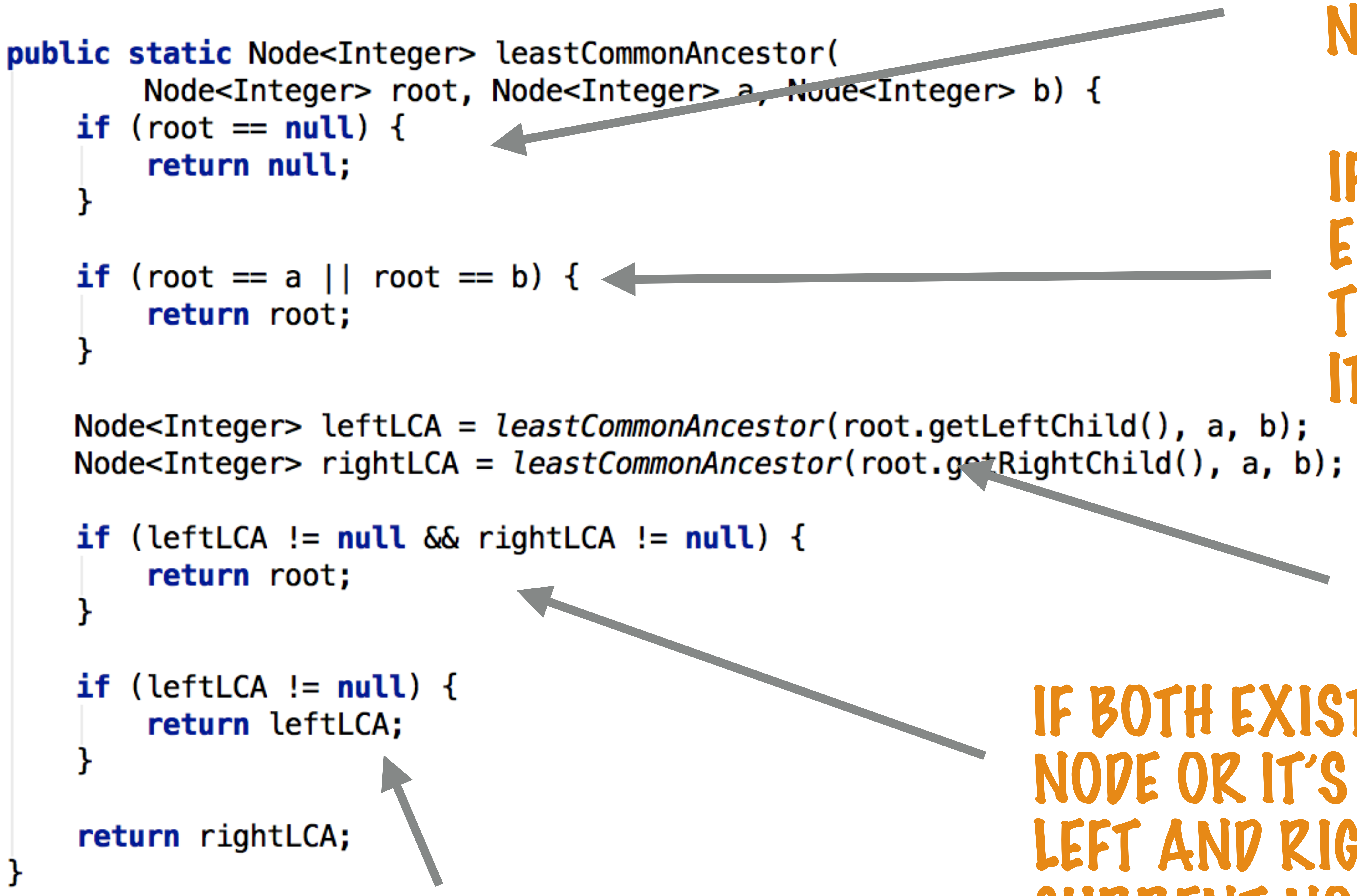
FIND THE LEAST COMMON ANCESTOR FOR 2 NODES



3 IS THE LEAST COMMON ANCESTOR FOR 8 AND 6.

NOTE THAT 1 IS ALSO A COMMON ANCESTOR BUT NOT THE LEAST COMMON ONE

LEAST COMMON ANCESTOR



```
public static Node<Integer> leastCommonAncestor(
        Node<Integer> root, Node<Integer> a, Node<Integer> b) {
    if (root == null) {
        return null;
    }

    if (root == a || root == b) {
        return root;
    }

    Node<Integer> leftLCA = leastCommonAncestor(root.getLeftChild(), a, b);
    Node<Integer> rightLCA = leastCommonAncestor(root.getRightChild(), a, b);

    if (leftLCA != null && rightLCA != null) {
        return root;
    }

    if (leftLCA != null) {
        return leftLCA;
    }

    return rightLCA;
}
```

IF WE ENCOUNTER A NULL ROOT NO ANCESTOR WAS FOUND

IF THE CURRENT ROOT IS EITHER OF THE TWO NODES THEN RETURN THE ROOT ITSELF

FIND THE LCA FOR THE LEFT AND RIGHT SUBTREES

IF BOTH EXIST IT MEANS - EITHER THE NODE OR IT'S ANCESTOR EXISTS IN THE LEFT AND RIGHT SUBTREE SO THE CURRENT NODE IS THE LCA

IF ONLY ONE OF THE COMMON ANCESTORS IS NON NULL RETURN THAT