# FIND THE MINIMUM VALUE IN A BINARY SEARCH TREE

# FIND THE MINIMUM VALUE IN A BINARY SEARCH TREE

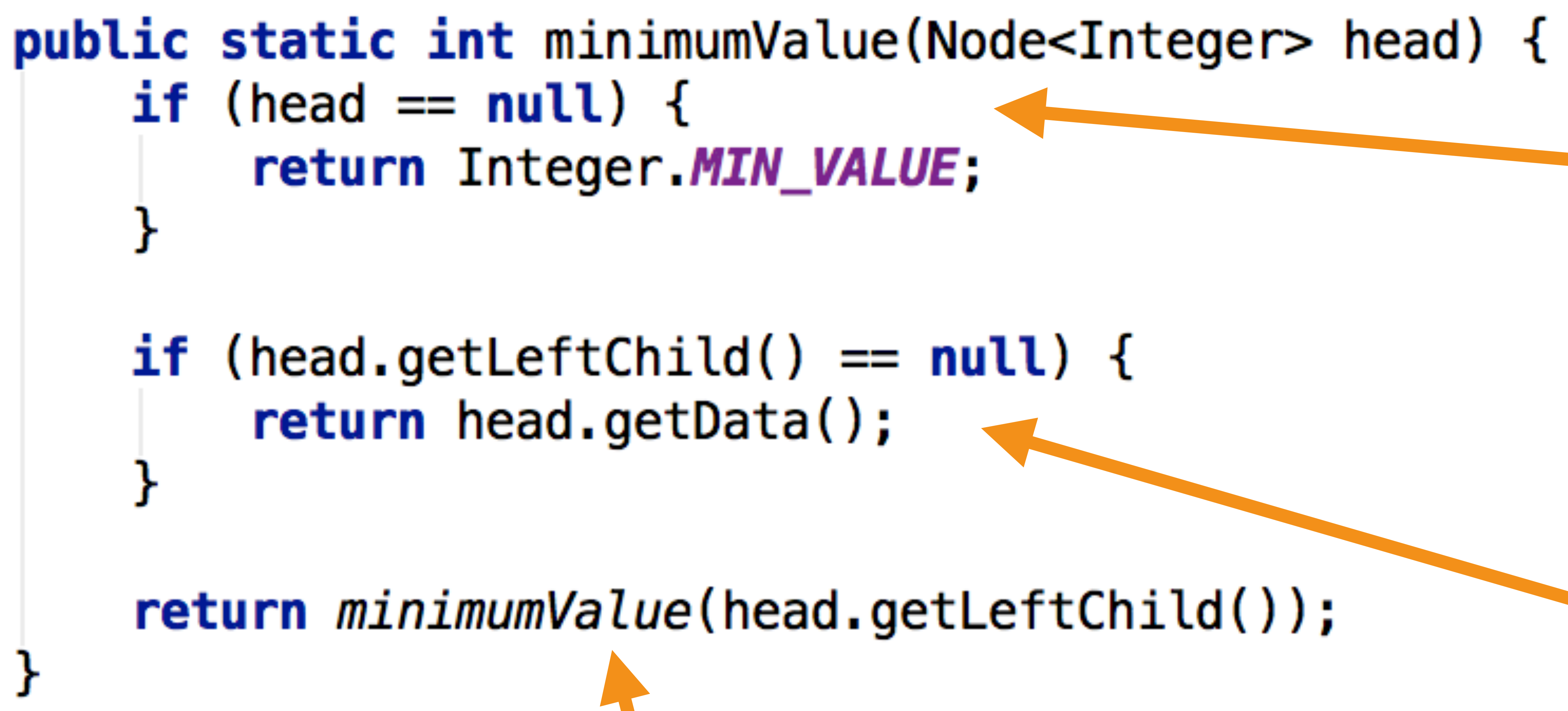
THE MINIMUM VALUE IN A BINARY SEARCH TREE CAN BE FOUND BY TRAVERSING THE LEFT SUBTREE OF EVERY NODE

FOR EVERY NODE, IT'S LEFT CHILD WILL HAVE A VALUE SMALLER THAN THE NODE'S VALUE

IF A NODE HAS NO LEFT CHILD

THAT IS THE NODE WITH THE SMALLEST VALUE - THE LEFT MOST LEAF NODE IN THE TREE

# MINIMUM VALUE IN A BST



```java
public static int minimumValue(Node<Integer> head) {
    if (head == null) {
        return Integer.MIN_VALUE;
    }

    if (head.getLeftChild() == null) {
        return head.getData();
    }

    return minimumValue(head.getLeftChild());
}
```

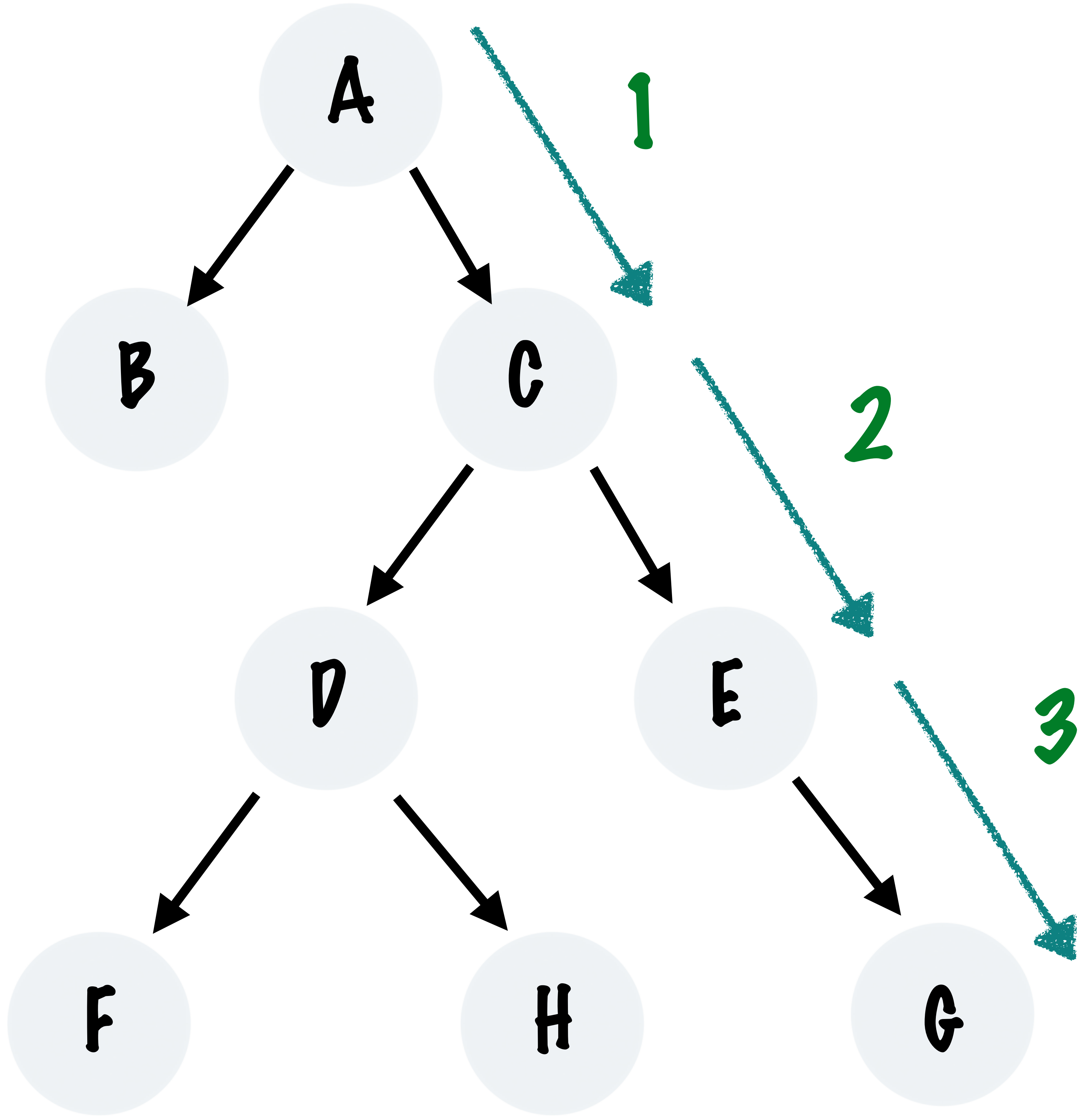
BASE CASE, IF THE HEAD IS NULL THEN THE TREE HAS NO NODES, RETURN THE MINIMUM INTEGER VALUE

FOLLOW THE LEFT CHILD FOR EVERY NODE, IF THE LEFT CHILD IS NULL THEN THIS IS THE MINIMUM VALUE NODE

RECURSE TILL A LEFT CHILD IS AVAILABLE

# FIND THE MAXIMUM DEPTH OF A BINARY TREE

# FIND THE MAXIMUM DEPTH OF A BINARY TREE

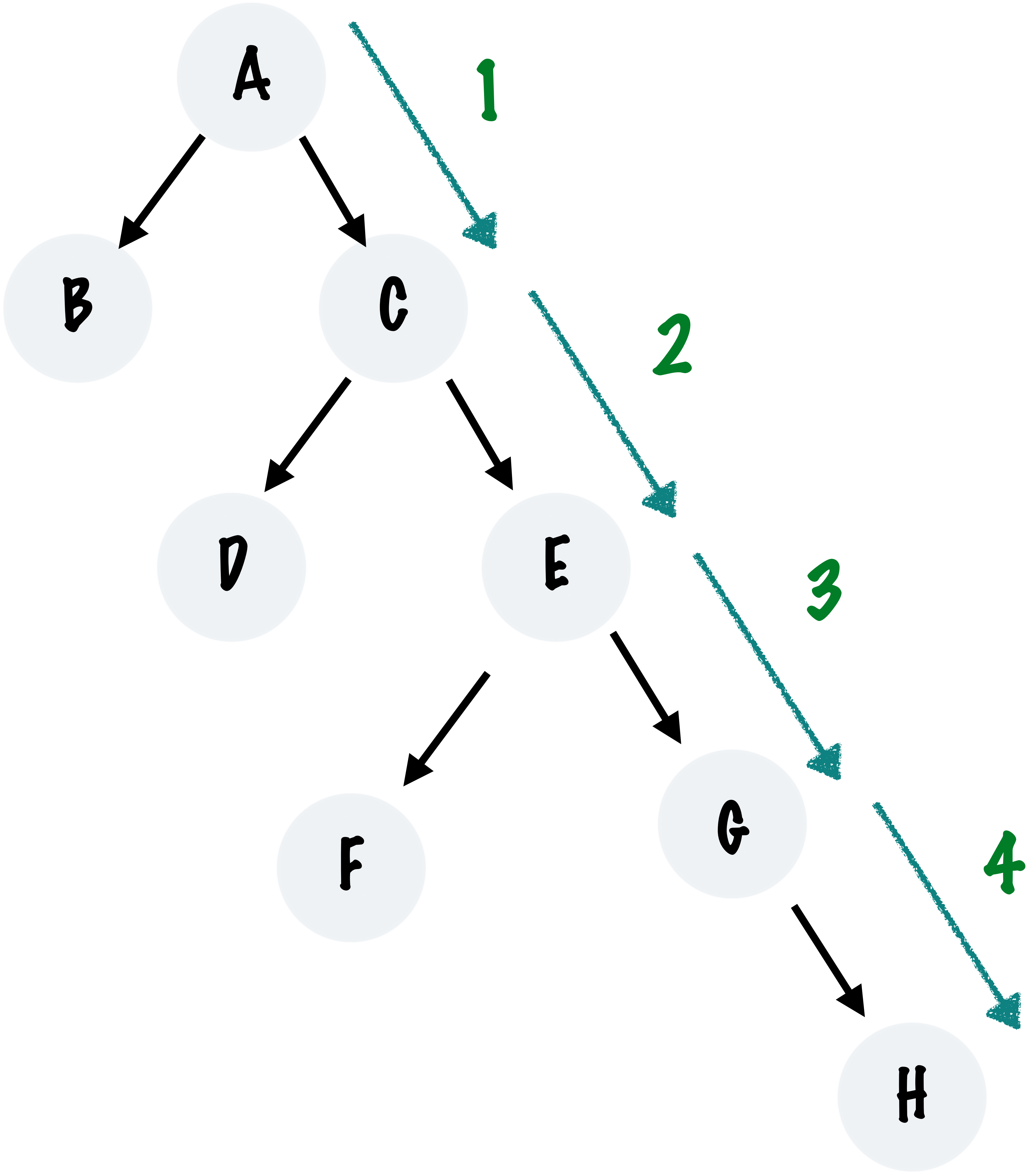


THE DEPTH OF A NODE IS IT'S DISTANCE FROM THE ROOT

THE MAX DEPTH WILL BE FURTHEST DISTANCE OF THE LEAF NODE FROM THE ROOT

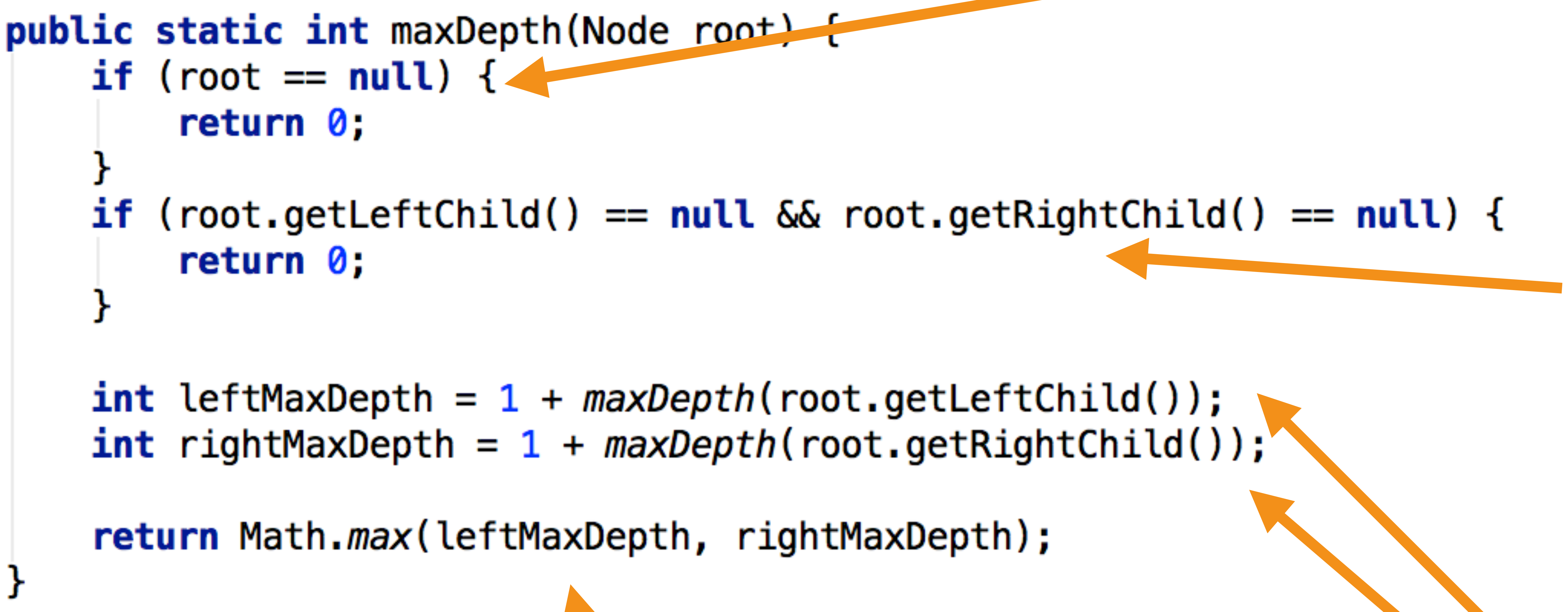
THIS TREE HAS A MAX DEPTH OF 3

# FIND THE MAXIMUM DEPTH OF A BINARY TREE



THIS TREE HAS A MAX DEPTH OF 4

# MAXIMUM DEPTH OF A BINARY TREE



```
public static int maxDepth(Node root) {
    if (root == null) {
        return 0;
    }
    if (root.getLeftChild() == null && root.getRightChild() == null) {
        return 0;
    }

    int leftMaxDepth = 1 + maxDepth(root.getLeftChild());
    int rightMaxDepth = 1 + maxDepth(root.getRightChild());

    return Math.max(leftMaxDepth, rightMaxDepth);
}
```

BASE CASE, IF THE ROOT IS NULL THEN THE TREE HAS NO NODES, THE MAX DEPTH IS 0

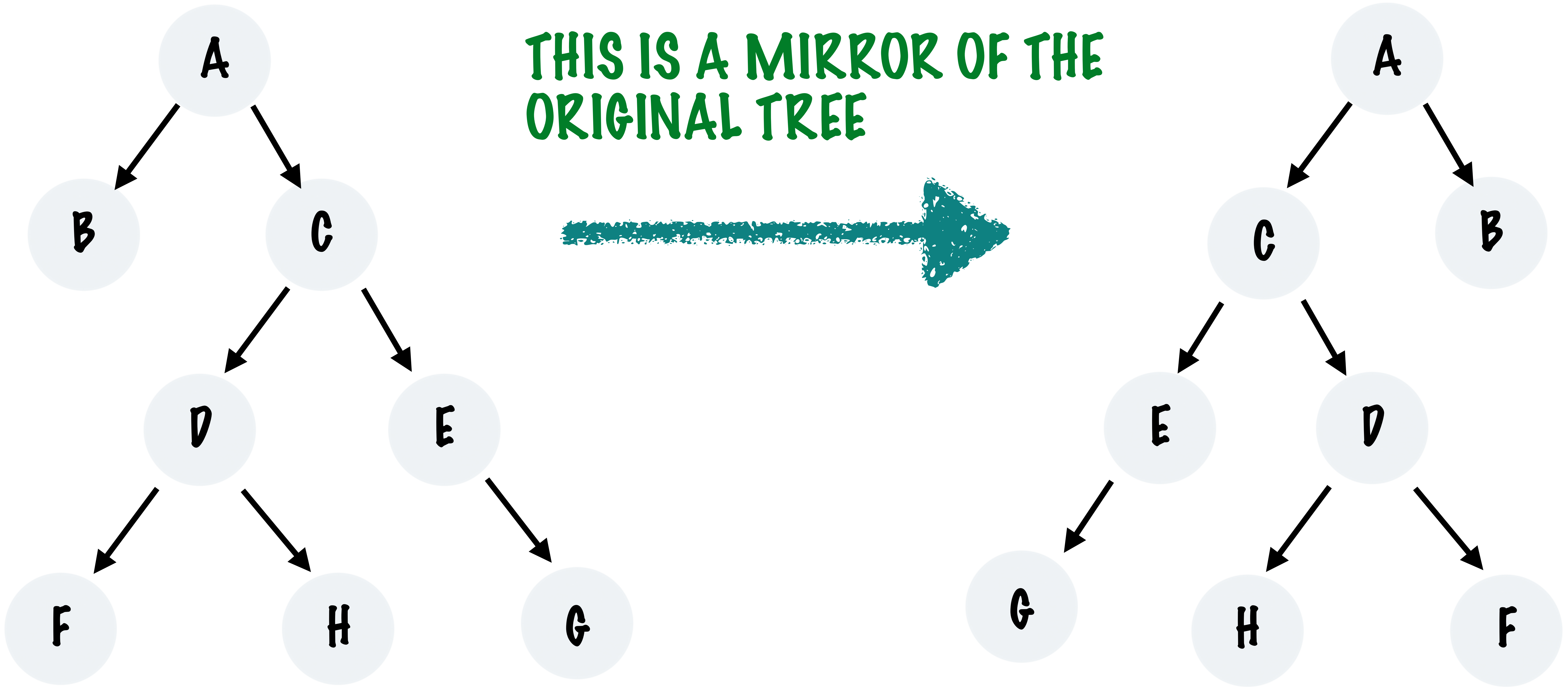
IF BOTH LEFT AND RIGHT CHILD OF THE NODE IS NULL THEN THIS IS A LEAF AND HAS A DEPTH OF 0

FIND THE MAX DEPTH ON THE LEFT AND RIGHT SUB-TREES - ADD 1 TO ACCOUNT FOR THE CURRENT DEPTH OF THE TREE

FIND THE MAX DEPTH BETWEEN THE LEFT AND RIGHT SUB-TREES

# MIRROR A BINARY TREE

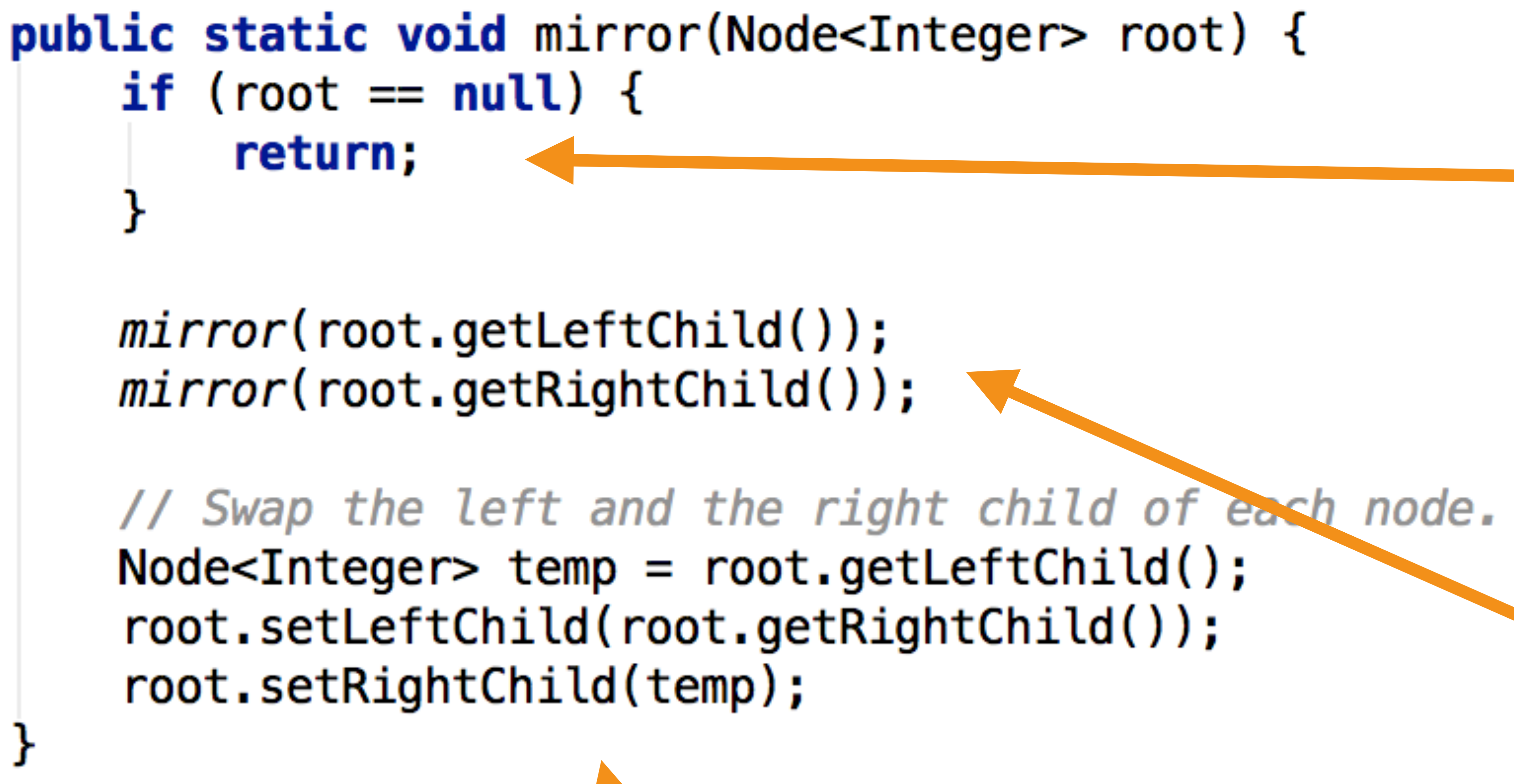
# MIRROR A BINARY TREE



THIS IS A MIRROR OF THE ORIGINAL TREE

EVERY LEFT CHILD IS NOW A RIGHT CHILD AND VICE VERSA

# MIRROR A BINARY TREE



```
public static void mirror(Node<Integer> root) {
    if (root == null) {
        return;
    }

    mirror(root.getLeftChild());
    mirror(root.getRightChild());

    // Swap the left and the right child of each node.
    Node<Integer> temp = root.getLeftChild();
    root.setLeftChild(root.getRightChild());
    root.setRightChild(temp);
}
```

BASE CASE, IF THE HEAD IS NULL THEN THE TREE HAS NO NODES, THERE IS NOTHING TO MIRROR

CALL MIRROR RECURSIVELY ON EVERY NODE IN THE LEFT AND RIGHT SUB-TREES

SWAP THE LEFT AND RIGHT CHILDREN OF THIS NODE