```
Codeium: Refactor | Explain
3  public class DiameterBinaryTree {
4      //Given a binary tree, you need to compute the length
5      // of the diameter of the tree.
6      //The diameter of a binary tree is the length of
7      // the longest path between any two nodes in a tree.
8      //This path may or may not pass through the root.
9      //Example: Given a binary tree
10     //          1
11     //         / \
12     //        2   3
13     //       / \
14     //      4   5
15     //Return 3, which is the length of the path [4,2,1,3] or [5,2,1,3].
16
       Codeium: Refactor | Explain
17     static class TreeNode {
18         int val;
19         TreeNode left;
20         TreeNode right;
21         TreeNode(int x) {
22             val = x;
23         }
24     }
25
       Codeium: Refactor | Explain | Generate Javadoc | ×
26     public static void main(String[] args) {
27         TreeNode root = new TreeNode(1);
28         root.left = new TreeNode(2);
29         root.right = new TreeNode(3);
30         root.left.left = new TreeNode(4);
31         root.left.right = new TreeNode(5);
32         System.out.println(diameterOfBinaryTree(root));
33     }
34
35     static int max = 0;
```

```java
    static int max = 0;
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int diameterOfBinaryTree(TreeNode root) {
        if (root == null) {
            return 0;
        }
        maxDepth(root);
        return max;
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }

        int left = maxDepth(root.left);
        int right = maxDepth(root.right);

        max = Math.max(max, left + right);

        return 1 + Math.max(left, right);
    }

    //Solve by Dynamic Programming
    //Approach: Dynamic Programming
    Codeium: Refactor | Explain | ×
    public static int diameterOfBinaryTreeDP(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int[] max = new int[1];
        maxDepth(root, max);
        return max[0];
    }

```

```
Codeium: Refactor | Explain | Generate Javadoc | ×
68      public static int maxDepth(TreeNode root, int[] max) {
69          if (root == null) {
70              return 0;
71          }
72
73          int left = maxDepth(root.left, max);
74          int right = maxDepth(root.right, max);
75
76          max[0] = Math.max(max[0], left + right);
77
78          return 1 + Math.max(left, right);
79      }
80
81
82  }
83
```

```
.DiameterBinaryTree
3
```

```java
Codeium: Refactor | Explain
public class DiameterNarrayTree {
    //Given an N-ary tree, find the diameter of the tree.
    //The diameter of an N-ary tree is the length of the
    // longest path between any two nodes in a tree.
    //This path may or may not pass through the root.
    //Example: Given a binary tree
    //          1
    //         /|\ \
    //        2 3 4 5
    //       /|\
    //      6 7 8
    //Return 4, which is the length of the path [6,2,1,3,4,8,7,2].

    Codeium: Refactor | Explain
    static class TreeNode {
        int val;
        TreeNode[] children;
        TreeNode(int x) {
            val = x;
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.children = new TreeNode[4];
        root.children[0] = new TreeNode(2);
        root.children[1] = new TreeNode(3);
        root.children[2] = new TreeNode(4);
        root.children[3] = new TreeNode(5);
        root.children[0].children = new TreeNode[3];
        root.children[0].children[0] = new TreeNode(6);
        root.children[0].children[1] = new TreeNode(7);
        root.children[0].children[2] = new TreeNode(8);
        System.out.println(diameterOfNaryTreeDP(root));
    }

    static int max = 0;
```

```java

    static int max = 0;
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int diameterOfNaryTree(TreeNode root) {
        if (root == null) {
            return 0;
        }
        maxDepth(root);
        return max;
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }

        int firstMax = 0;
        int secondMax = 0;
        for (TreeNode child : root.children) {
            int depth = maxDepth(child);
            if (depth > firstMax) {
                secondMax = firstMax;
                firstMax = depth;
            } else if (depth > secondMax) {
                secondMax = depth;
            }
        }

        max = Math.max(max, firstMax + secondMax);

        return 1 + firstMax;
    }

```

```java

    //Solve by Dynamic Programming
    //Approach: Dynamic Programming
    Codeium: Refactor | Explain | ×
    public static int diameterOfNaryTreeDP(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int[] max = new int[1];
        maxDepth(root, max);
        return max[0];
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int maxDepth(TreeNode root, int[] max) {
        if (root == null) {
            return 0;
        }

        int firstMax = 0;
        int secondMax = 0;
        for (TreeNode child : root.children) {
            int depth = maxDepth(child, max);
            if (depth > firstMax) {
                secondMax = firstMax;
                firstMax = depth;
            } else if (depth > secondMax) {
                secondMax = depth;
            }
        }

        max[0] = Math.max(max[0], firstMax + secondMax);

        return 1 + firstMax;
    }

}

```

```java
public class MaximumPathSumAnyNodeToAnyNode {
    //Given a binary tree, find the maximum path sum.
    // The path may start and end at any node in the tree.
    //Example: Given the below binary tree
    //         1
    //        / \
    //       2   3
    //Return 6. The path 2 -> 1 -> 3 gives the maximum
    // path sum.

    Codeium: Refactor | Explain
    static class TreeNode {
        int val;
        TreeNode left;
        TreeNode right;
        TreeNode(int x) {
            val = x;
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        System.out.println(maxPathSum(root));
    }

    static int max = Integer.MIN_VALUE;
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int maxPathSum(TreeNode root) {
        if (root == null) {
            return 0;
        }
        maxPathSumUtil(root);
        return max;
    }

```

```
Codeium: Refactor | Explain | Generate Javadoc | ×
38      public static int maxPathSumUtil(TreeNode root) {
39          if (root == null) {
40              return 0;
41          }
42
43          int left = Math.max(0, maxPathSumUtil(root.left));
44          int right = Math.max(0, maxPathSumUtil(root.right));
45
46          max = Math.max(max, left + right + root.val);
47
48          return Math.max(left, right) + root.val;
49      }
50
51      //Approach: Dynamic Programming
Codeium: Refactor | Explain | ×
52      public static int maxPathSumDP(TreeNode root) {
53          if (root == null) {
54              return 0;
55          }
56
57          int[] max = new int[1];
58          max[0] = Integer.MIN_VALUE;
59          maxPathSumUtilDP(root, max);
60          return max[0];
61      }
62
Codeium: Refactor | Explain | Generate Javadoc | ×
63      public static int maxPathSumUtilDP(TreeNode root, int[] max) {
64          if (root == null) {
65              return 0;
66          }
67
68          int left = Math.max(0, maxPathSumUtilDP(root.left, max));
69          int right = Math.max(0, maxPathSumUtilDP(root.right, max));
70
71          max[0] = Math.max(max[0], left + right + root.val);
72
73          return Math.max(left, right) + root.val;
74      }
75
76  }
77
```

```
.MaximumPathSumAnyNodeToAnyNode
6
```

```
Codeium: Refactor | Explain
public class MaximumPathSumLeafToLeaf {
    //If a binary tree is given, how to find Maximum path sum
    // between two leaves of binary tree.
    //
    //All should be numbers
    //The maximum sum path may or may not go through root.
    //For example, in the following binary tree, the maximum sum
    // is 27(3 + 6 + 9 + 0 -1 + 10). Expected time complexity is O(n)
    // where n is the number of nodes in the given Binary Tree.
    //         -15
    //         / \
    //        5   6
    //       / \ / \
    //     -8  1 3  9
    //    /  / \   / \
    //   2  6   0 6  4
    //      / \
    //     0   10
    //Approach: Dynamic Programming

    Codeium: Refactor | Explain
    static class TreeNode {
        int val;
        TreeNode left;
        TreeNode right;
        TreeNode(int x) {
            val = x;
        }
    }

```

```java
Codeium: Refactor | Explain | Generate Javadoc | ×
public static void main(String[] args) {
    TreeNode root = new TreeNode(-15);
    root.left = new TreeNode(5);
    root.right = new TreeNode(6);
    root.left.left = new TreeNode(-8);
    root.left.right = new TreeNode(1);
    root.right.left = new TreeNode(3);
    root.right.right = new TreeNode(9);
    root.left.left.left = new TreeNode(2);
    root.left.right.left = new TreeNode(6);
    root.left.right.right = new TreeNode(0);
    root.right.right.left = new TreeNode(6);
    root.right.right.right = new TreeNode(4);
    root.left.right.left.left = new TreeNode(0);
    root.left.right.left.right = new TreeNode(10);
    System.out.println(maxPathSum(root));
}

static int max = Integer.MIN_VALUE;
Codeium: Refactor | Explain | Generate Javadoc | ×
public static int maxPathSum(TreeNode root) {
    if (root == null) {
        return 0;
    }
    maxPathSumUtil(root);
    return max;
}

Codeium: Refactor | Explain | Generate Javadoc | ×
public static int maxPathSumUtil(TreeNode root) {
    if (root == null) {
        return 0;
    }

    int left = Math.max(0, maxPathSumUtil(root.left));
    int right = Math.max(0, maxPathSumUtil(root.right));

    max = Math.max(max, left + right + root.val);

    return Math.max(left, right) + root.val;
}
```

```
    }

    //Approach: Dynamic Programming
    Codeium: Refactor | Explain | ×
    public static int maxPathSumDP(TreeNode root) {
        if (root == null) {
            return 0;
        }

        int[] max = new int[1];
        max[0] = Integer.MIN_VALUE;
        maxPathSumUtilDP(root, max);
        return max[0];
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int maxPathSumUtilDP(TreeNode root, int[] max) {
        if (root == null) {
            return 0;
        }

        int left = Math.max(0, maxPathSumUtilDP(root.left, max));
        int right = Math.max(0, maxPathSumUtilDP(root.right, max));

        max[0] = Math.max(max[0], left + right + root.val);

        return Math.max(left, right) + root.val;
    }
}

```

```
.MaximumPathSumLeafToLeaf
28
```