**day17\_107856406\_dsdipt\_sudipto\_10july2025**

**Employee Code:** 107856406

**Login ID:** dsdipt

**Email :** dsdipt@amazon.com

**Name:** Sudipto Das

**Date:** 10 July 2025 (Day 17)

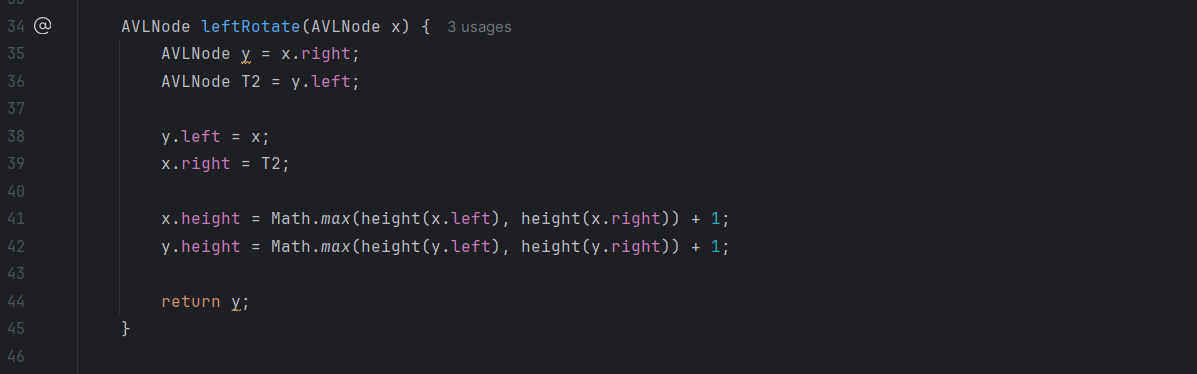
### ***Task 1: Write Algo for AVL Tree***

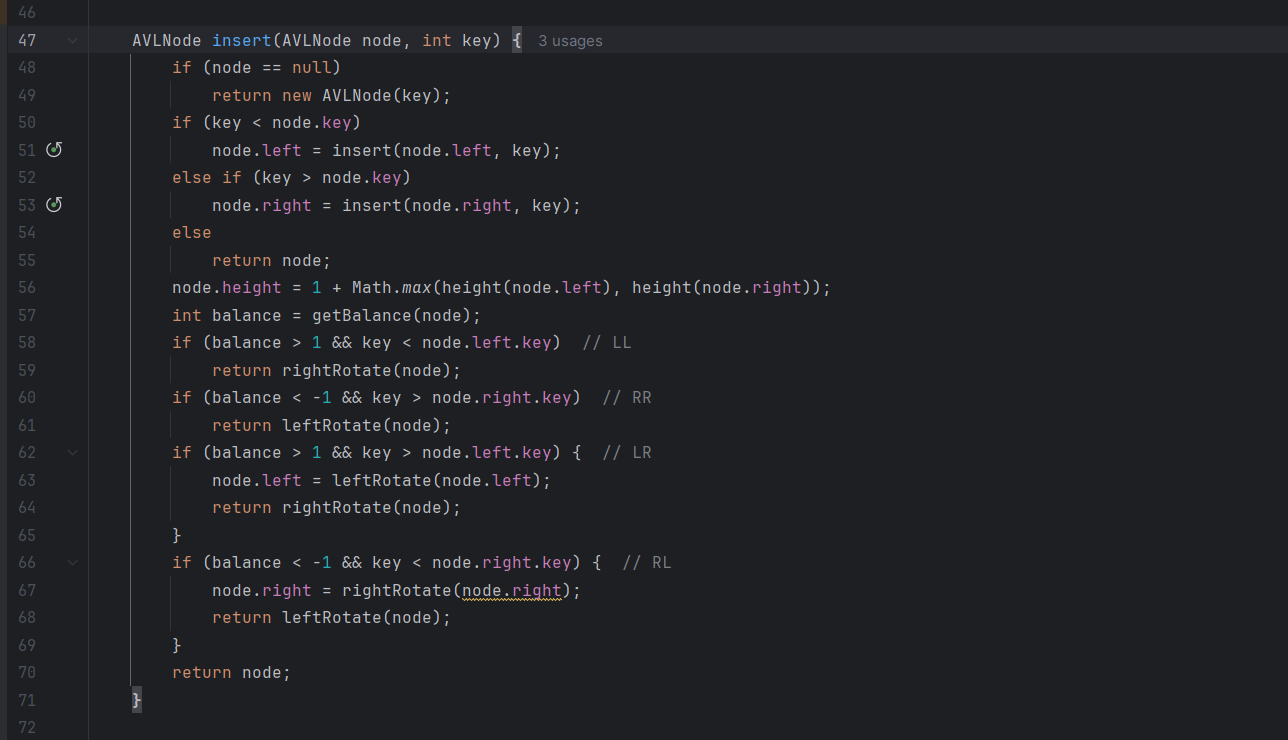
**Function: insert(node, key)**

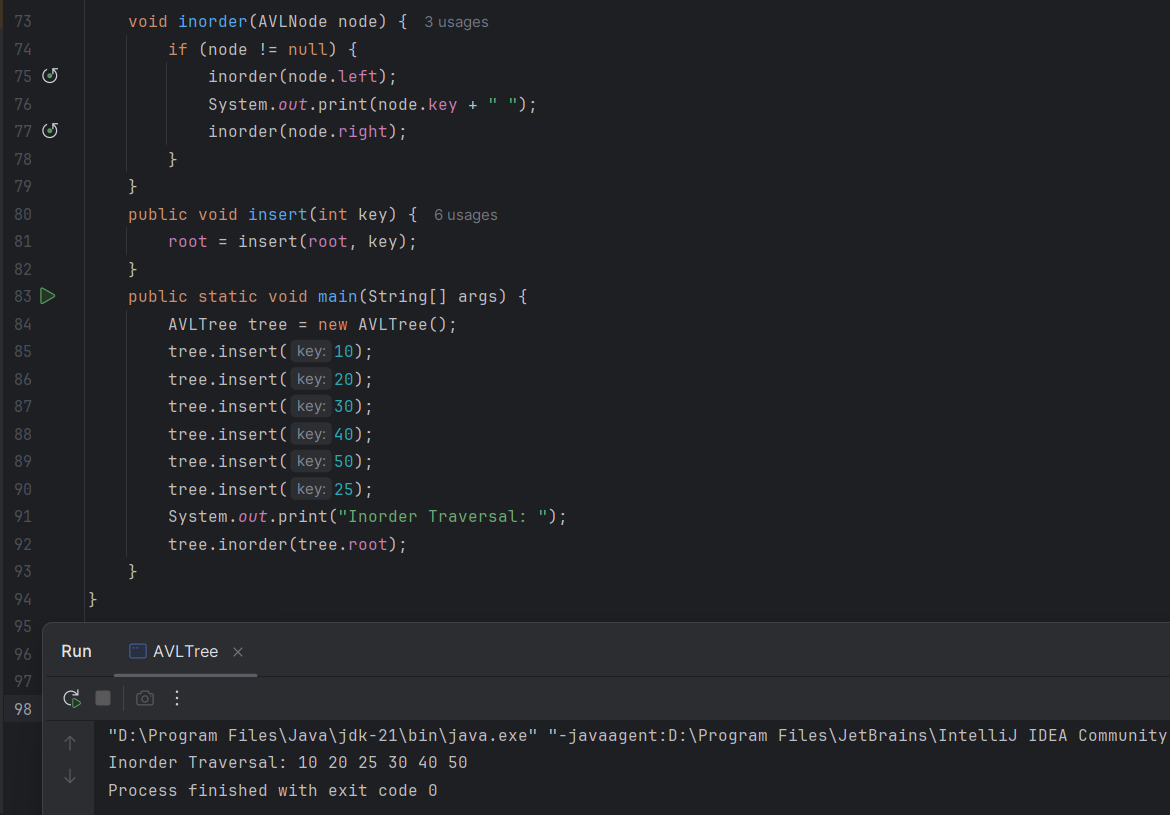
1. **If** node is null →  
    ➤ Create and return new node.
2. **If** key < node.key →  
    ➤ Insert in the left subtree.
3. **If** key > node.key →  
    ➤ Insert in the right subtree.
4. **Update height** of current node.
5. **Calculate balance** = height(left) - height(right)
6. **Balance using rotations**:
   * **Left-Left (LL)** → Right Rotate
   * **Right-Right (RR)** → Left Rotate
   * **Left-Right (LR)** → Left Rotate left child, then Right Rotate
   * **Right-Left (RL)** → Right Rotate right child, then Left Rotate
7. **Return the balanced node**

***Task 2: Implement AVL Tree***









***Task 3: Write Algo for Red Black Tree***

Algorithm: Insert(RBT, value)

Input: A red-black tree RBT and a value to insert

Output: Updated RBT maintaining red-black properties

1. Insert the node like in a regular Binary Search Tree (BST):

- Create a new node with the value

- Color it RED

- Place it in the correct position (left or right of parent)

2. Fix any violations of red-black properties:

- While the parent of the new node is RED:

- Let P = parent(node), G = grandparent(node), U = uncle(node)

Case 1: Uncle is RED

- Color P and U as BLACK

- Color G as RED

- Move current node to G (go up the tree)

Case 2: Uncle is BLACK or NULL

- If node is inner child (left-right or right-left):

- Rotate parent in opposite direction (to convert to outer child)

- Then rotate grandparent in opposite direction of node

- Swap colors of parent and grandparent

3. Ensure the root is always BLACK

***Task 4: Implement Red Black Tree***

