**Task 1: Balanced Binary Tree Check**

**Write a function to check if a given binary tree is balanced. A balanced tree is one where the height of two subtrees of any node never differs by more than one.**

**class TreeNode {**

**int val;**

**TreeNode left;**

**TreeNode right;**

**TreeNode(int x) {**

**val = x;**

**}**

**}**

**public class BinaryTreeBalance {**

**// Helper function to check the height of the tree**

**private static int checkHeight(TreeNode node) {**

**if (node == null) {**

**return 0; // Base case: The height of an empty tree is 0**

**}**

**// Recursively get the height of the left and right subtrees**

**int leftHeight = checkHeight(node.left);**

**int rightHeight = checkHeight(node.right);**

**// If the left or right subtree is unbalanced, propagate the error (-1)**

**if (leftHeight == -1 || rightHeight == -1) {**

**return -1;**

**}**

**// If the current node is unbalanced, return -1**

**if (Math.abs(leftHeight - rightHeight) > 1) {**

**return -1;**

**}**

**// Otherwise, return the height of the tree rooted at the current node**

**return Math.max(leftHeight, rightHeight) + 1;**

**}**

**// Main function to check if the tree is balanced**

**public static boolean isBalanced(TreeNode root) {**

**return checkHeight(root) != -1;**

**}**

**public static void main(String[] args) {**

**// Create a sample balanced binary tree:**

**// 1**

**// / \**

**// 2 3**

**// / \**

**// 4 5**

**TreeNode root = new TreeNode(1);**

**root.left = new TreeNode(2);**

**root.right = new TreeNode(3);**

**root.left.left = new TreeNode(4);**

**root.left.right = new TreeNode(5);**

**System.out.println("Is the tree balanced? " + isBalanced(root)); // Should print true**

**// Create an unbalanced tree:**

**// 1**

**// /**

**// 2**

**// /**

**// 3**

**root = new TreeNode(1);**

**root.left = new TreeNode(2);**

**root.left.left = new TreeNode(3);**

**System.out.println("Is the tree balanced? " + isBalanced(root)); // Should print false**

**}**

**}**