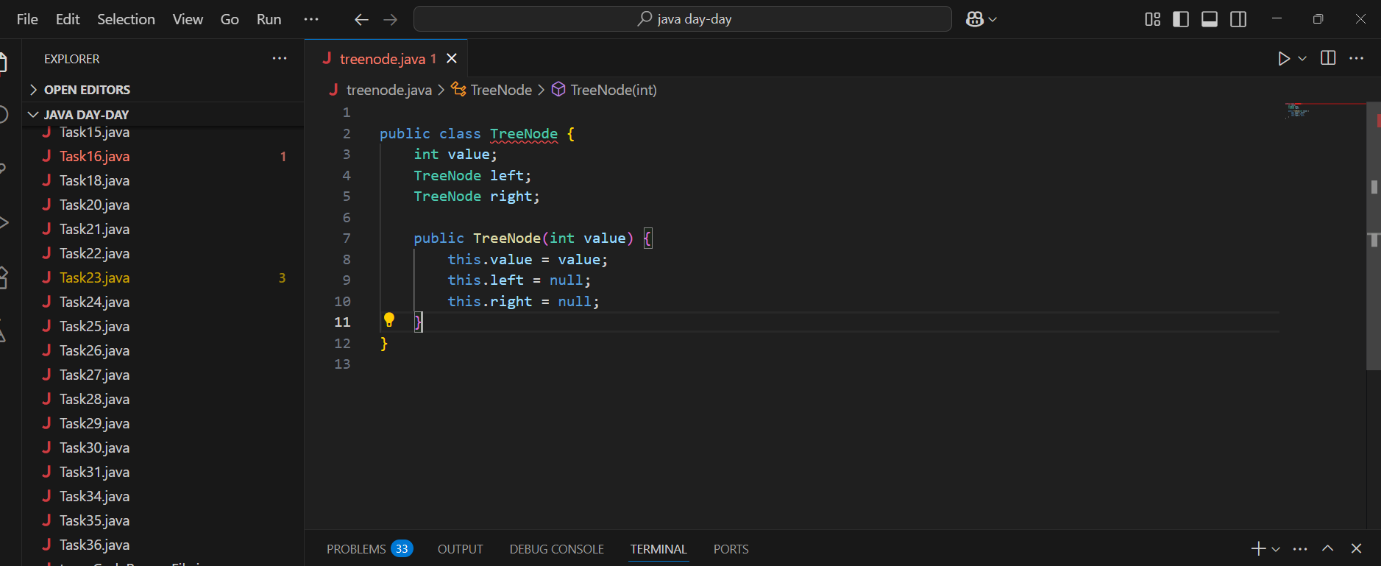
Day 15 - 03rd July 2025

Name: Aravind Kasanagottu

Id: mvsnarav

Task 1:

Create a node  for a tree and include a constructor (empty)



Hint 👍

A node which consists of 1 data part and 2 refs  ( 1 Left ref and another right ref)

Solution:

class TreeNode {

    int value;

    TreeNode left, right;

    TreeNode(int item) {

        value = item;

        left = right = null;

    }

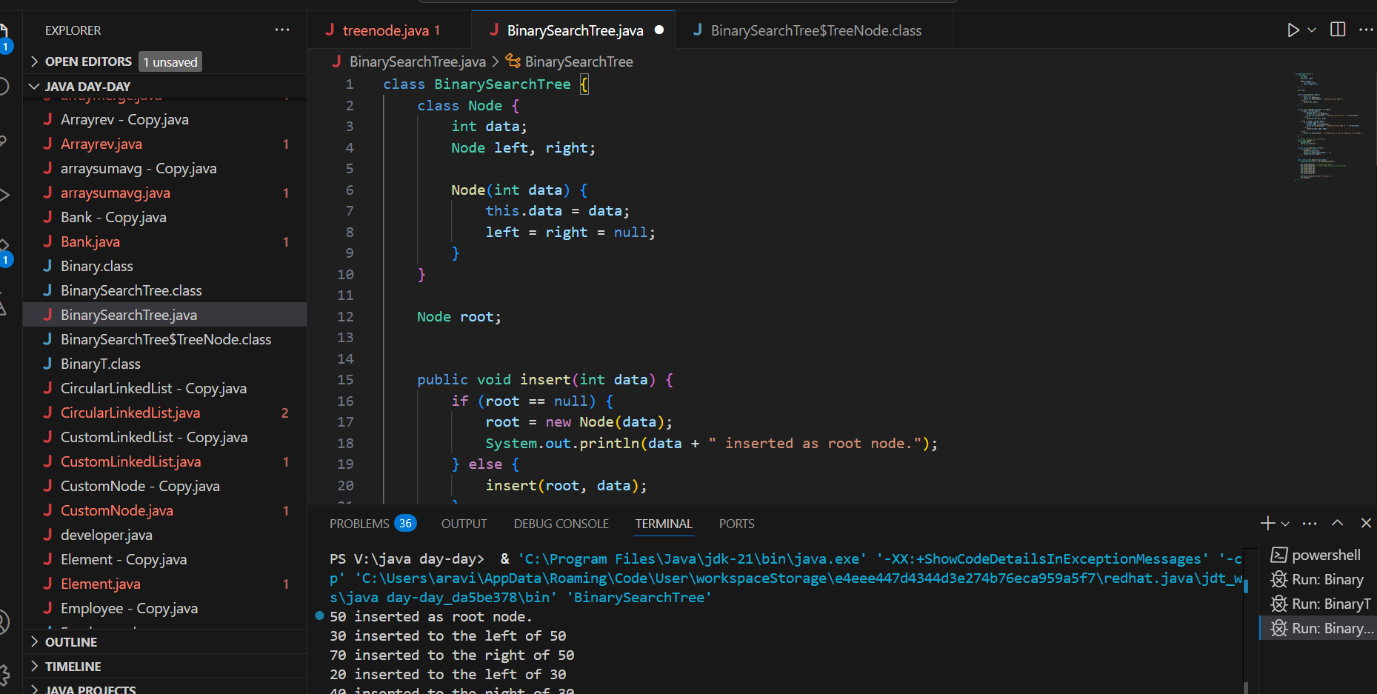
}

Task 2:

Create a class named Binarty Search tree in which you have 2 insert operations

1 insert —----> for inserting if the tree is empty

1 insert —----> for inserting if the tree is 1 or more nodes



    TreeNode insertVal(TreeNode node, int value) {

        if (node == null) {

            node = new TreeNode(value);

            return node;

        }

        if (value < node.value) {

            node.left = insertVal(node.left, value);

        } else if (value > node.value) {

            node.right = insertVal(node.right, value);

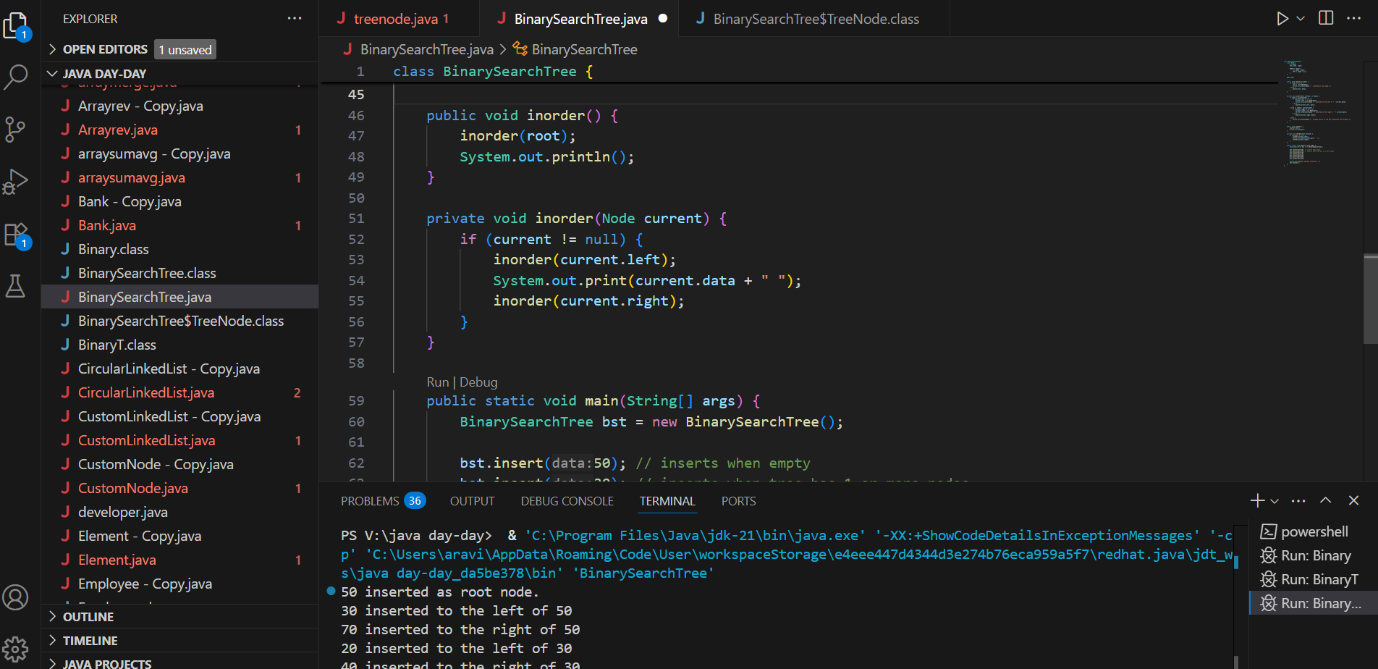
        }

        return node;

    }

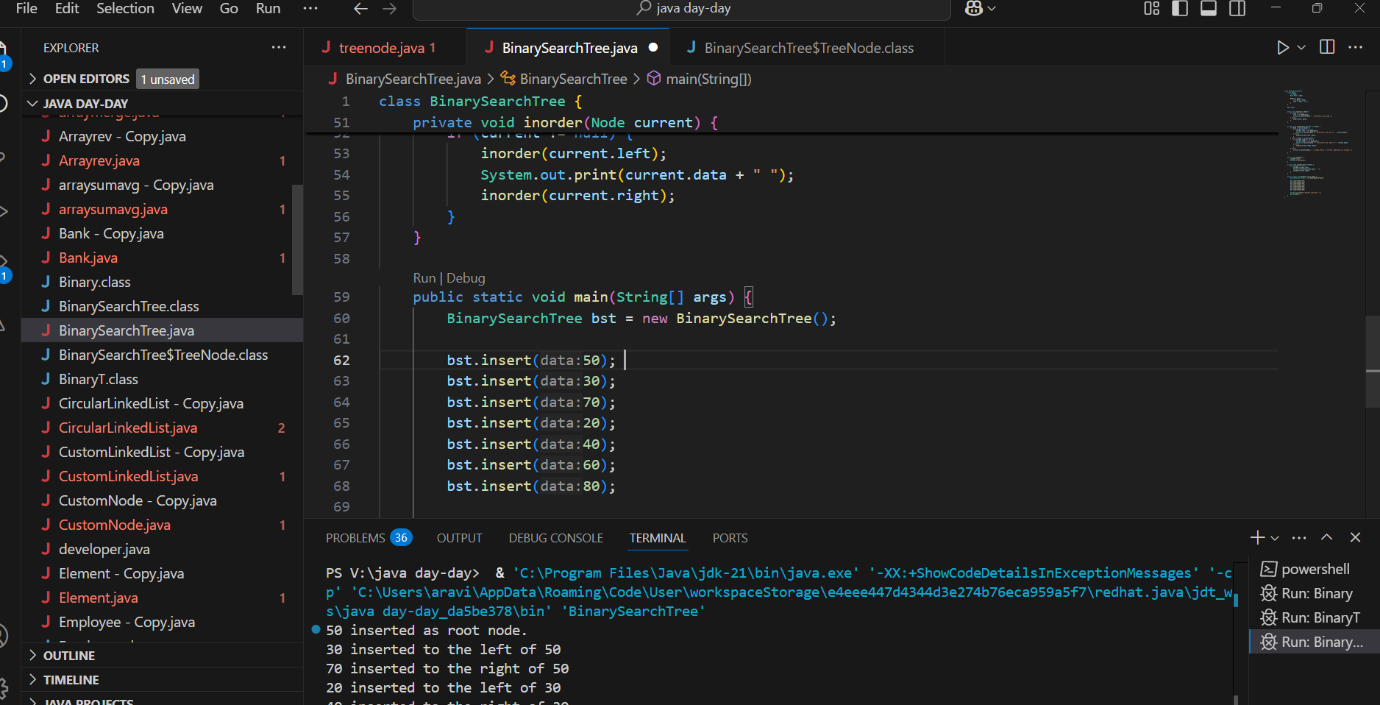
Task 3:

Ionorder travel of the above code snippets from task 1 and Task 2



Task 4:

Create  a main method Task 1, 2 and 3



class TreeNode {

    int value;

    TreeNode left, right;

    TreeNode(int item) {

        value = item;

        left = right = null;

    }

}

class BinarySearchTreeOp {

    TreeNode root;

    void insert(int value) {

        root = insertVal(root, value);

    }

    TreeNode insertVal(TreeNode node, int value) {

        if (node == null) {

            node = new TreeNode(value);

            return node;

        }

        if (value < node.value) {

            node.left = insertVal(node.left, value);

        } else if (value > node.value) {

            node.right = insertVal(node.right, value);

        }

        return node;

    }

    void inorder() {

        inorderVal(root);

    }

    void inorderVal(TreeNode node) {

        if (node != null) {

            inorderVal(node.left);

            System.out.print(node.value + " ");

            inorderVal(node.right);

        }

    }

}

public class BinarySearchTree {

    public static void main(String[] args) {

        BinarySearchTreeOp bstobj = new BinarySearchTreeOp();

bstobj.insert(10);

bstobj.insert(50);

bstobj.insert(40);

bstobj.insert(70);

bstobj.insert(5);

Sytem.out.println("here is the code for in order traversal of Binary search tree ");

bstobj.inorder();

    }

}

Task 5:

Applications of Trees

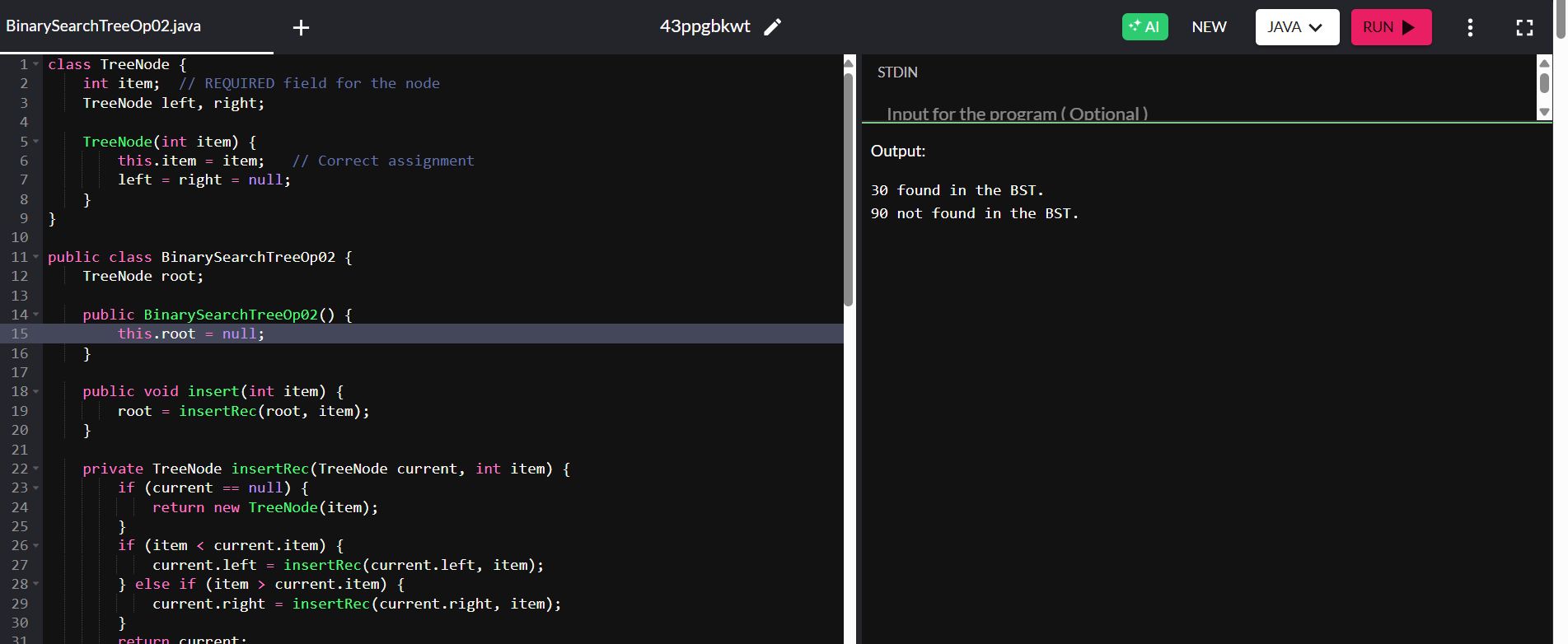
Applications of Trees:

1. Trees represent hierarchical data like file systems and organization charts.
2. They are used in database indexing (B-Trees, B+ Trees) for fast searching.
3. Trees help in expression parsing in compilers and in Huffman coding for compression.
4. Tries are used for routing and auto-completion in searching.
5. Trees are also used in AI for decision trees and game trees for taking optimal decisions.

Task 6:

 Create  a binary search operation on tree

Hint:



Task 7:

15.20 to 15.28

Solution 👍

class TreeNode {

    int item;

    TreeNode left, right;

    TreeNode(int item) {

        item = item;

        left = right = null;

    }

}

class BinarySearchTreeOp02 {

    TreeNode root;

    public BinarySearchTreeOp02() {

        this.root = null;

    }

    public TreeNode search(int key) {

        TreeNode current = root;

        while (current != null) { // key 30    current 50 == root

            if (key == current.item) {

                return current;

            } else if (key < current.item) { // key 80    current 50 == root

                current = current.left;

            } else {

                current = current.right;

            }

        }

        return null;

    }

}

Task 8:

Types of binary trees:

Types of binary trees:  
A full binary tree has every node with 0 or 2 children.  
A complete binary tree has all levels filled except the last, filled left to right.  
A perfect binary tree has all internal nodes with 2 children and leaves at the same level.  
A skewed binary tree has nodes only on one side, either all left or all right.  
A balanced binary tree keeps the height minimal for faster operations.

===============================================================================================================================================

Graphs

===============================================================================================================================================

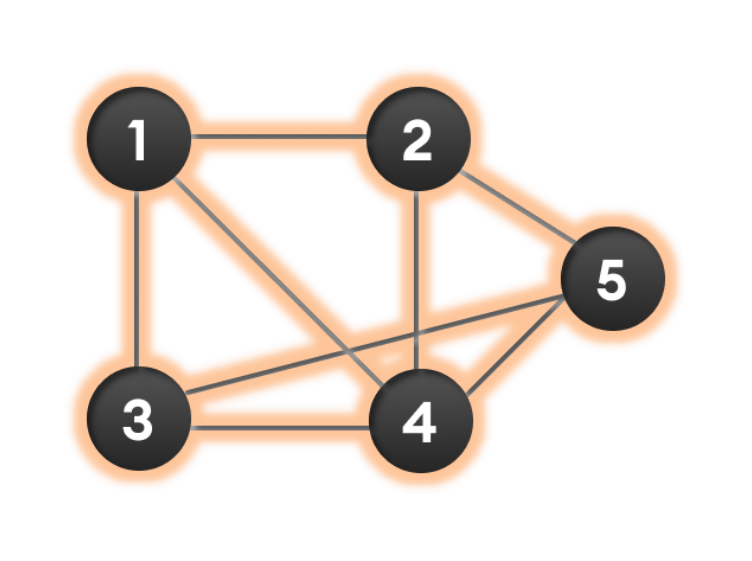
Task 9:

Graphs are used to represent networks like social media connections and computer networks.  
They are used in Google Maps for finding the shortest path.  
They are used in network flow for traffic and data routing.  
Graphs are also used in AI for state space and pathfinding problems.

Task 10:

Types of Graphs:

A **simple graph** has no loops or multiple edges.  
A **multigraph** allows multiple edges between the same vertices.  
A **directed graph (digraph)** has edges with direction.  
A **weighted graph** has edges with weights or costs.  
A **complete graph** has every pair of vertices connected directly.



Wap to display a graph edges .in the below order no od edges 8 and no of vertex 5

1 - 2

1 - 3

1 - 4

2 - 4

2 - 5

3 - 4

3 - 5

4 - 5

Hint:

Class Graph{

Class Edge{

Int start/src;

Int end/dest;

}

Int vertex;

Int edge;

}

Solution:

class Graph01 {

class Edge {

     int src, dest;

}

int vertices, edges;

Edge[] edge;

Graph(int vertices, int eges) {

this.vertices = vertices;

     this.edges = edges;

edge = new Edge[edges];

for(int i = 0; i < edges; i++) {

edge[i] = new Edge();

}

}

public static void main(String[] args) {

int noVertices = 5;

    int noEdges = 8;

     Graph gObj = new Graph(noVertices, noEdges);

gObj.edge[0].src = 1;

     gObj.edge[0].dest = 2;

    gObj.edge[1].src = 1;

     gObj.edge[1].dest = 3;

     gObj.edge[2].src = 1;

     gObj.edge[2].dest = 4;

gObj.edge[3].src = 2;

gObj.edge[3].dest = 4;

gObj.edge[4].src = 2;

gObj.edge[4].dest = 5;

gObj.edge[5].src = 3;

gObj.edge[5].dest = 4;

gObj.edge[6].src = 3;

gObj.edge[6].dest = 5;

gObj.edge[7].src = 4;

gObj.edge[7].dest = 5;

for(int i =0; i < noEdges; i++) {

System.out.println(gObj.edge[i].src+ " - " + gObj.edge[i].dest);

}

    }

}