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)]

Example :

class TreeNode {

int value;

TreeNode left, right;

TreeNode(int item) {

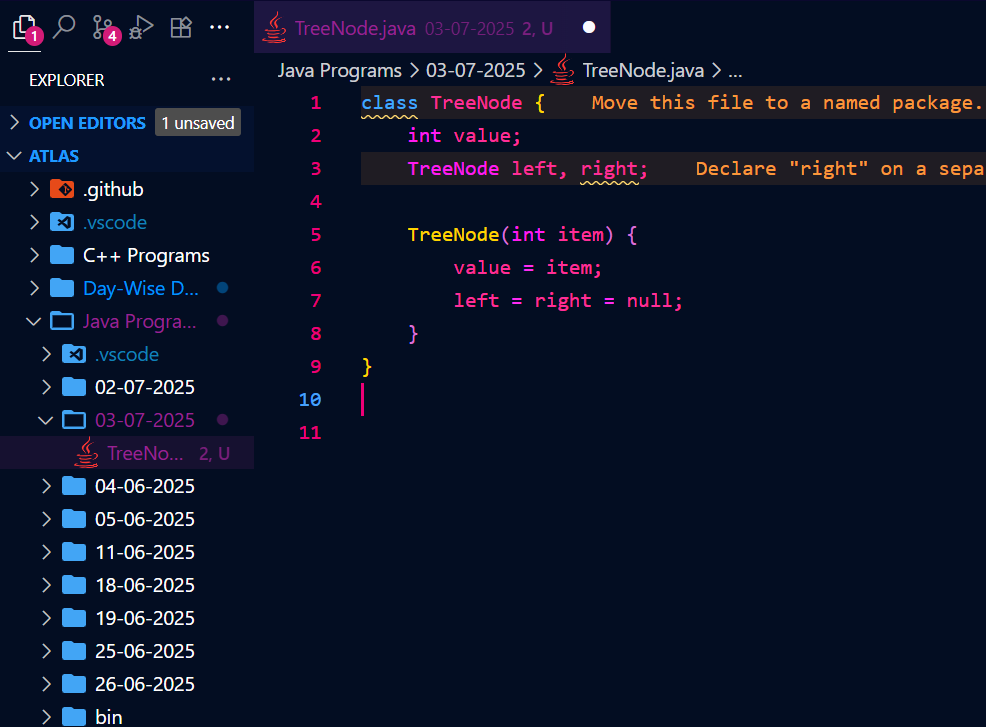
value = item;

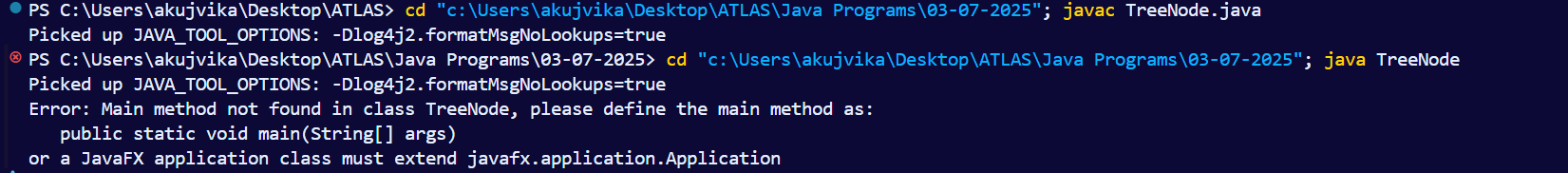
left = right = null;

}

}

Solution : This class represents a node in a binary tree. Each node contains an integer value and pointers to its left and right children.





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);

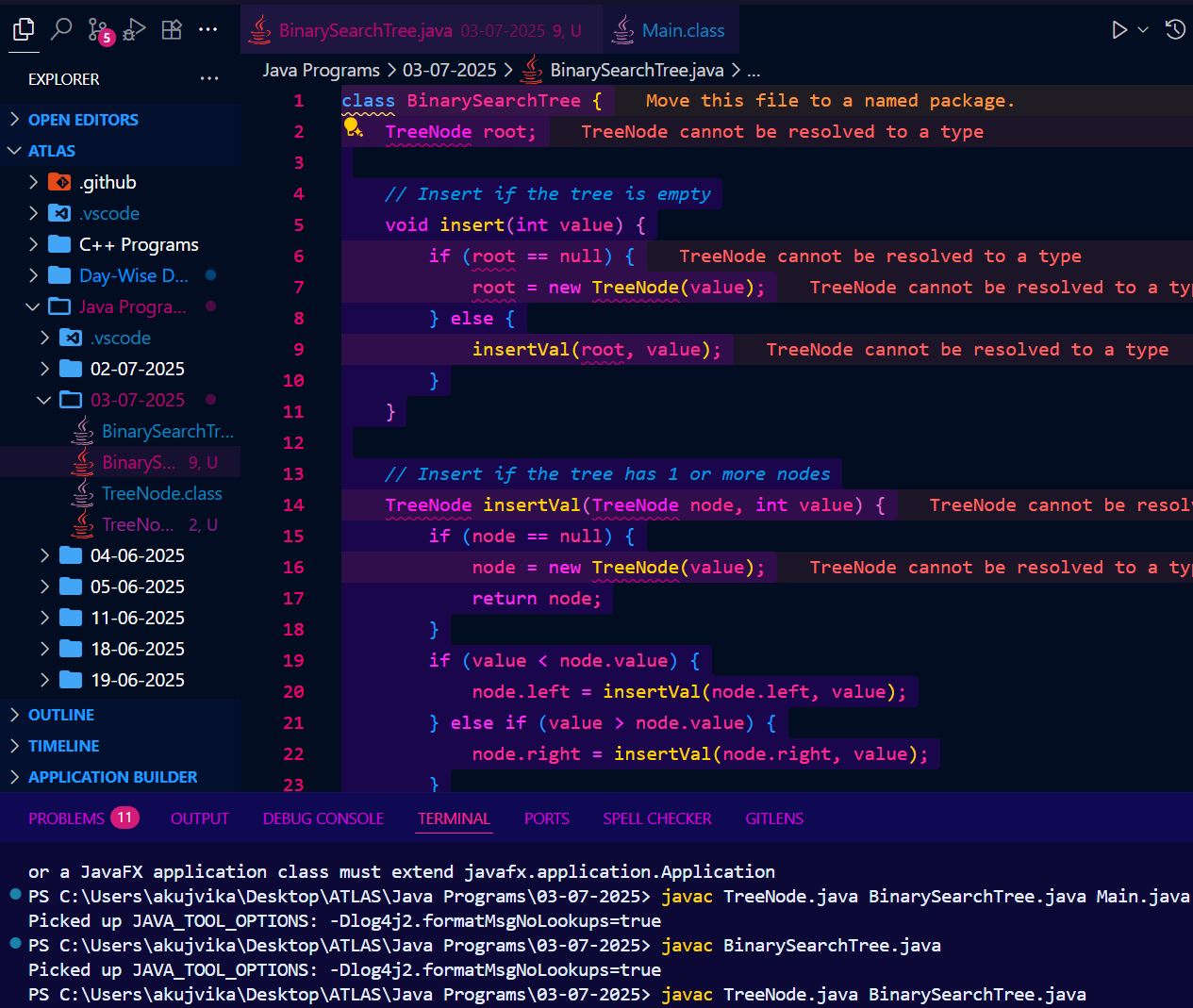
} else if (value > node.value) {

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

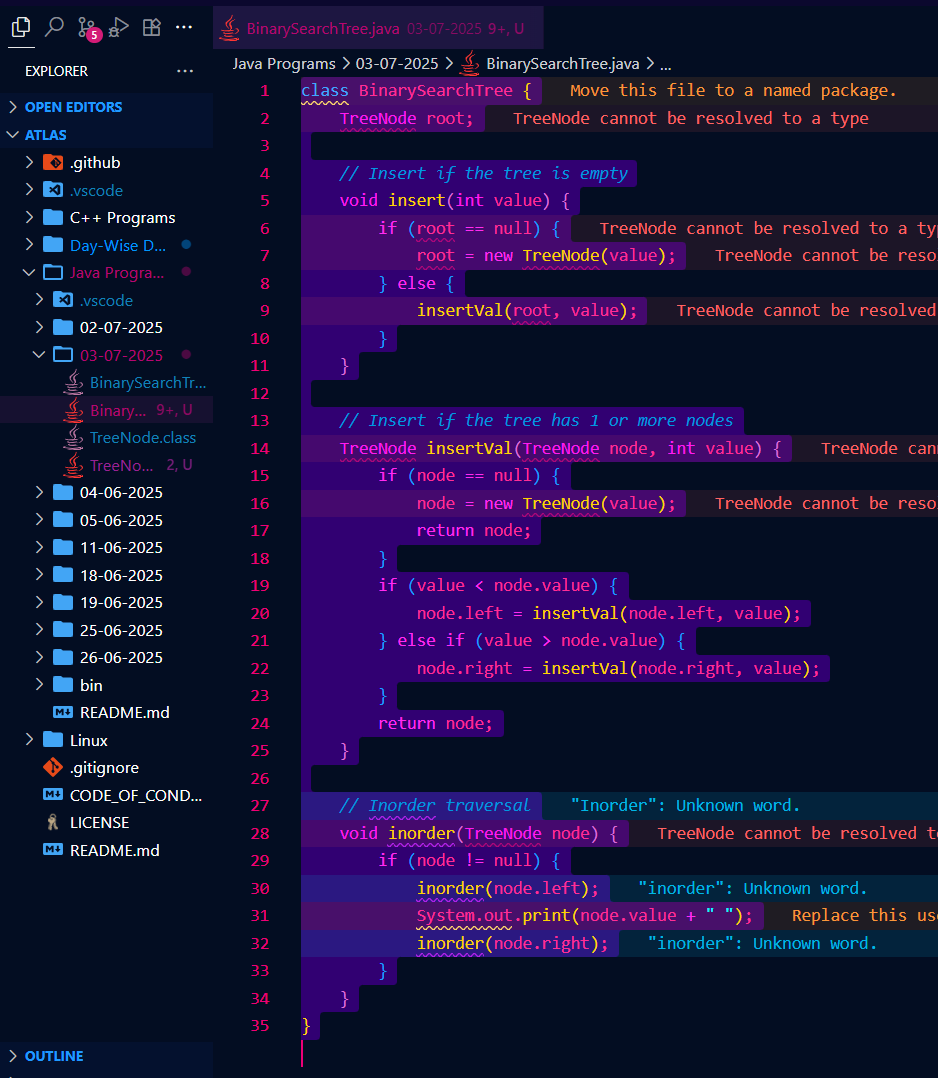
}

return node;

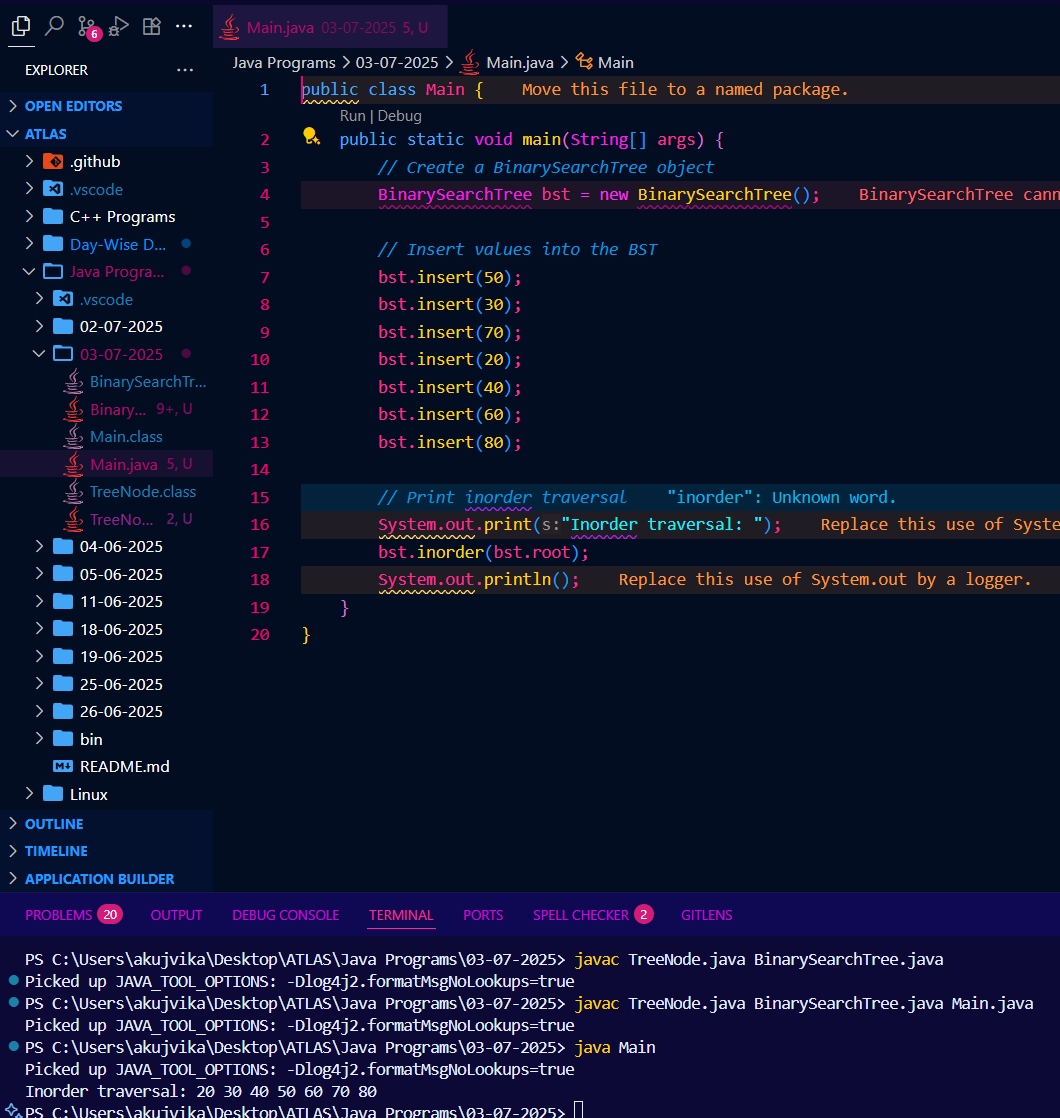
}

Solution : 

Task 3: Ionorder travel of the above code snippets from task 1 and Task 2

Solution : 

Task 4: Create a main method Task 1, 2 and 3 and run the code..

Solution : 

Task 5 : Few applications of Trees & Graphs.

Solution : Here are a few important applications of Trees and Graphs:

### Trees

1. \*\*Hierarchical Data Representation:\*\*

- File systems (folders and files)

- Organization charts

- XML/HTML document structure

2. \*\*Searching and Sorting:\*\*

- Binary Search Trees (BST) for fast lookup, insertion, and deletion

- Heaps for priority queues (used in scheduling, Dijkstra’s algorithm)

3. \*\*Expression Parsing:\*\*

- Abstract Syntax Trees (AST) in compilers and interpreters

4. \*\*Databases:\*\*

- B-trees and B+ trees for indexing and efficient data retrieval

### Graphs

1. \*\*Network Routing:\*\*

- Internet, telephone, and transport networks (finding shortest/optimal paths)

2. \*\*Social Networks:\*\*

- Modeling relationships and connections (friends, followers)

3. \*\*Recommendation Systems:\*\*

- Product/user/item relationships (collaborative filtering)

4. \*\*Dependency Resolution:\*\*

- Package managers, build systems (resolving dependencies)

5. \*\*Web Crawling/Search Engines:\*\*

- Representing and traversing the web (pages as nodes, links as edges)

Task 6: Create a binary search operation on tree

(Hint: Create a node , Class for binary search

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;

}

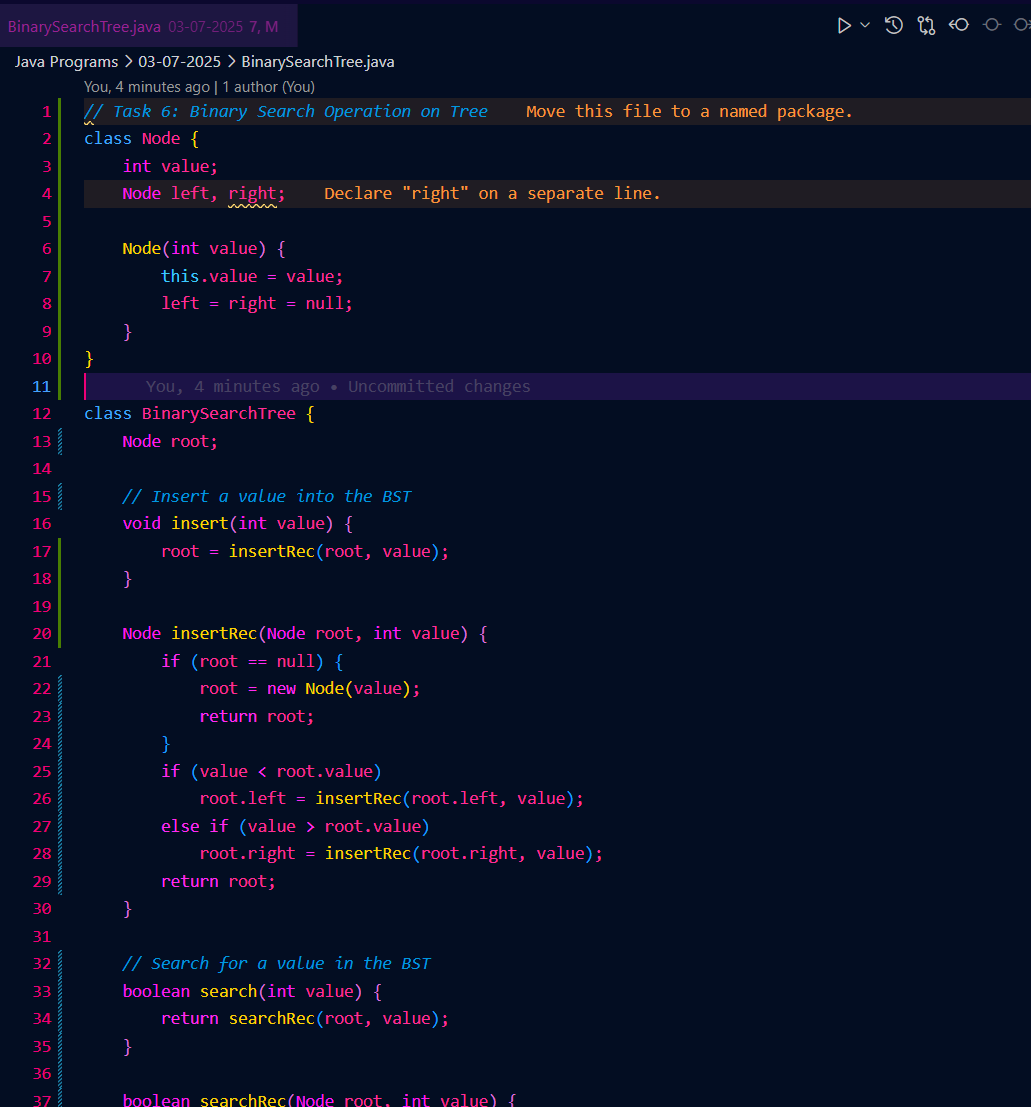
}

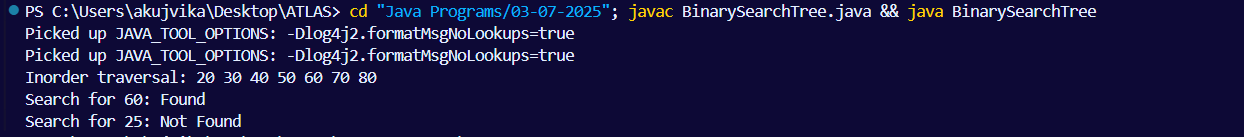
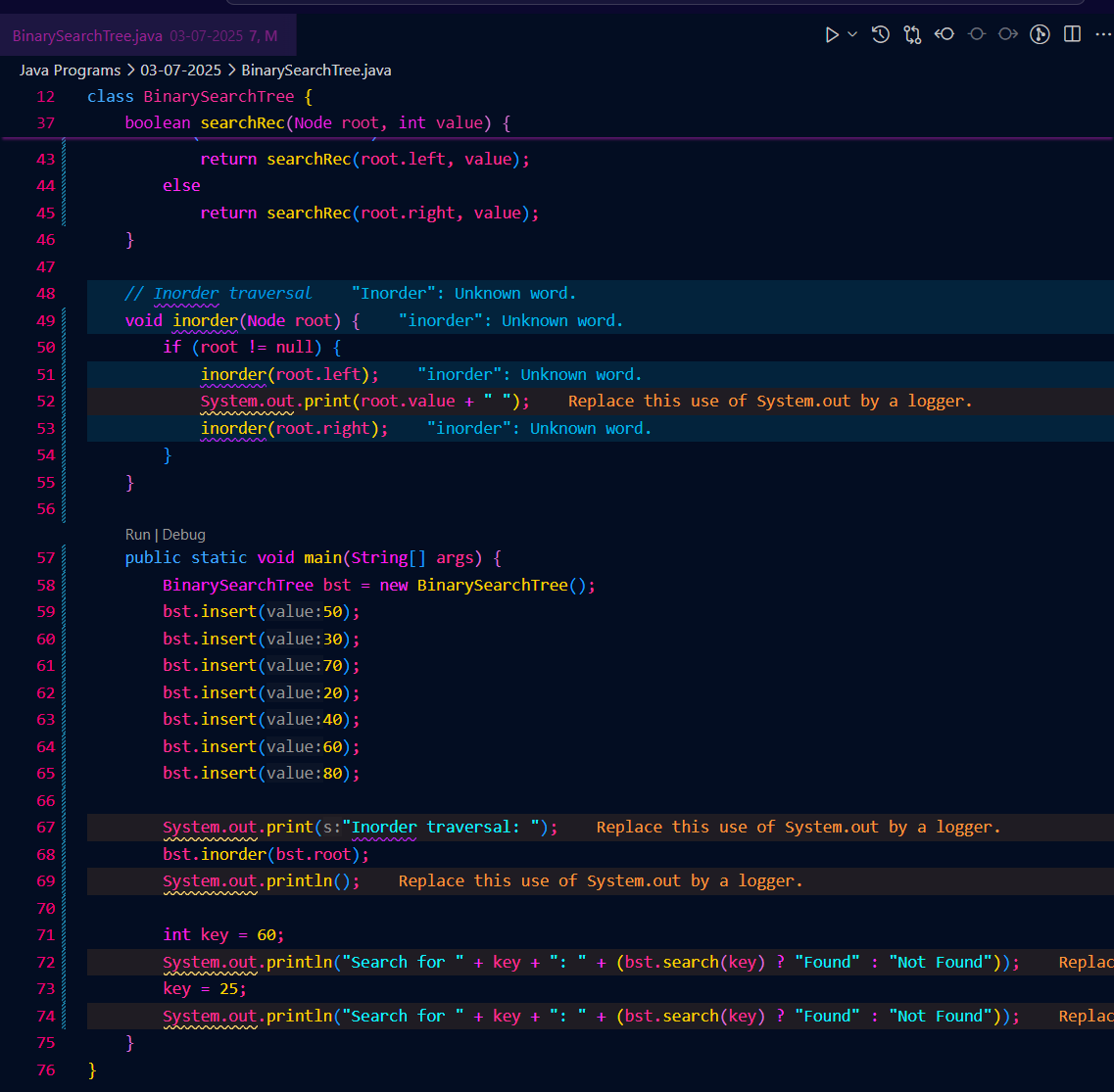
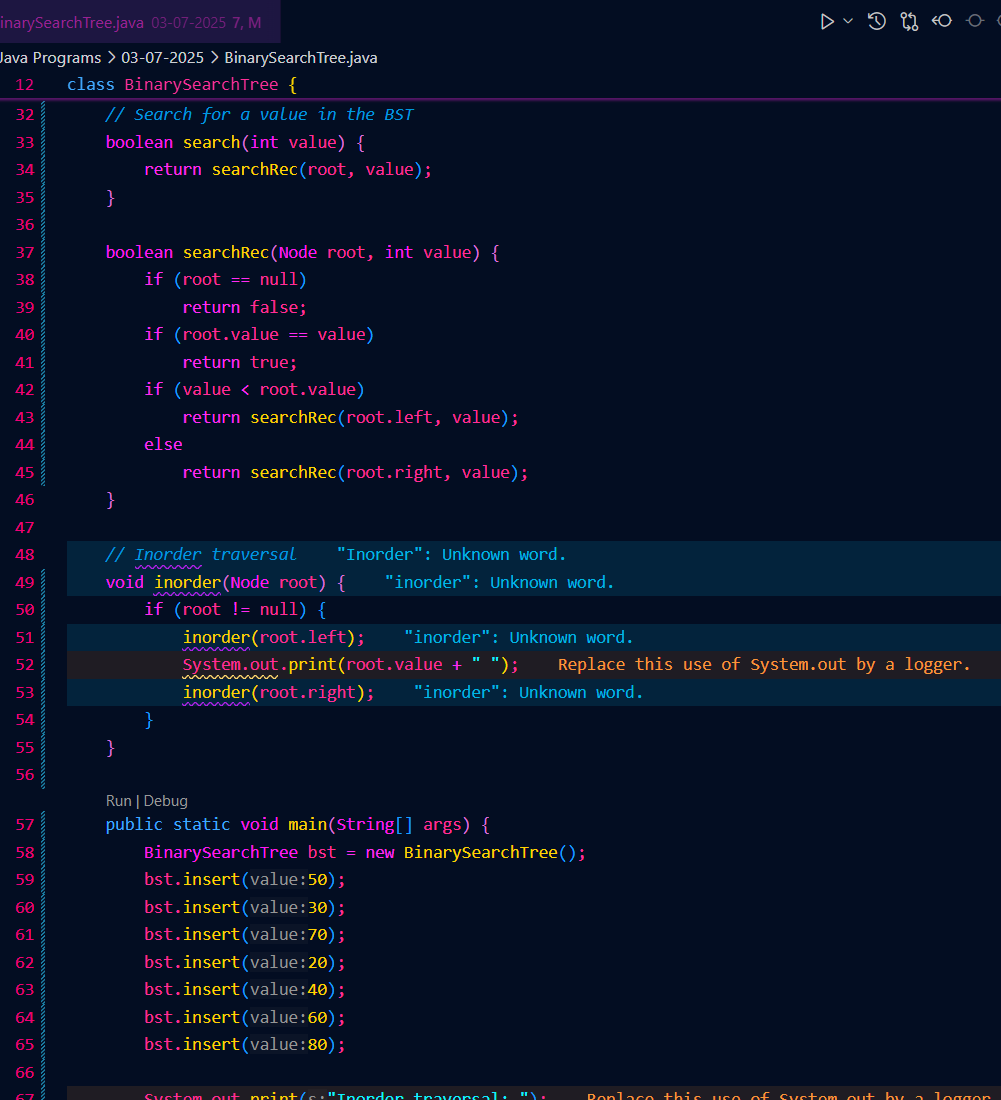
return null;

}

})

Solution :





Task 7: Deletion

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;

}

}

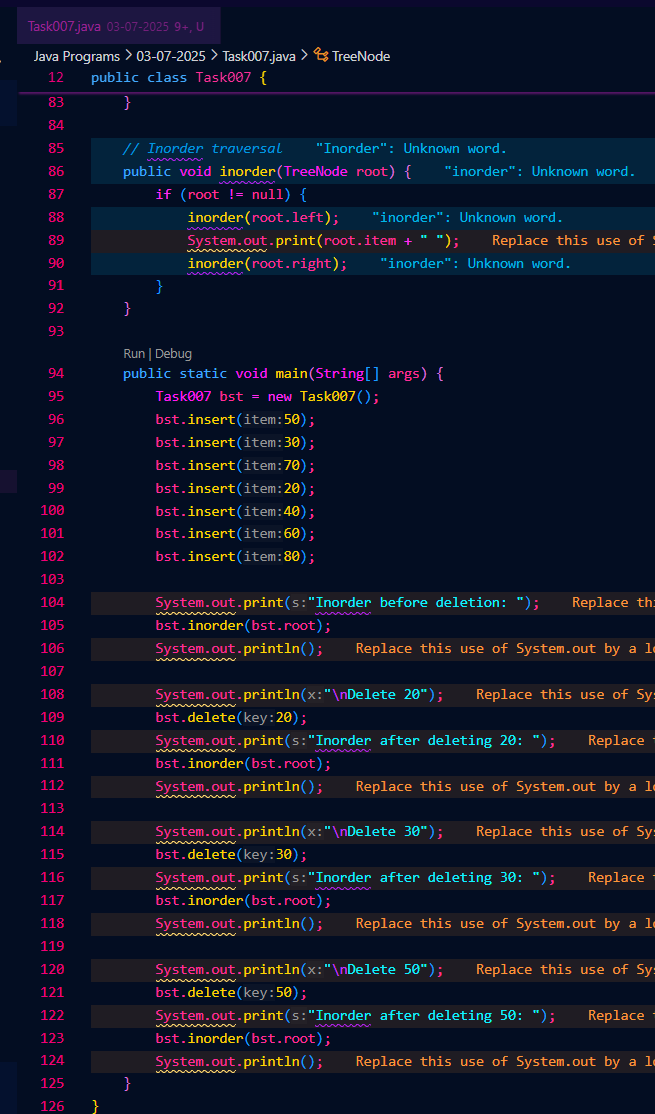
return null;

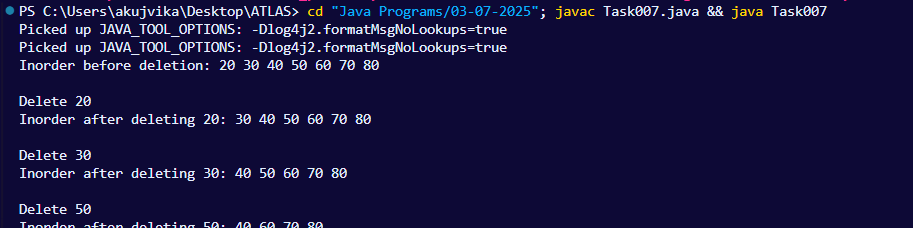
}

}

Solution : 







Task 8: Types of binary trees:

Solution :

Full Binary Tree: Every node has 0 or 2 children.

Perfect Binary Tree: All internal nodes have 2 children and all leaves are at the same level.

Complete Binary Tree: All levels are completely filled except possibly the last, which is filled from left to right.

Skewed Binary Tree: All nodes have only left or only right child (left-skewed or right-skewed).

Balanced Binary Tree: The height difference between left and right subtrees of any node is at most 1 (e.g., AVL tree).

Degenerate (or Pathological) Tree: Each parent node has only one child, making it look like a linked list.

Task 9: Applications of Graphs

Solution :

Social networks (modeling relationships)

Internet/web page links (web graph)

Network routing (finding shortest/optimal paths)

Recommendation systems (user/item relationships)

Scheduling and project planning (dependency graphs)

Maps and navigation (road networks)

Circuit design (electrical networks)

Search engines (web crawling)

Task 10: Types of Graphs

Solution :

Directed Graph (Digraph): Edges have a direction.

Undirected Graph: Edges have no direction.

Weighted Graph: Edges have weights/costs.

Unweighted Graph: Edges do not have weights.

Cyclic Graph: Contains at least one cycle.

Acyclic Graph: No cycles (e.g., DAG - Directed Acyclic Graph).

Connected Graph: There is a path between every pair of vertices.

Disconnected Graph: Not all vertices are connected.

Complete Graph: Every pair of vertices is connected by an edge.

Sparse/Dense Graph: Based on the ratio of edges to vertices.

Task 11 : 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;

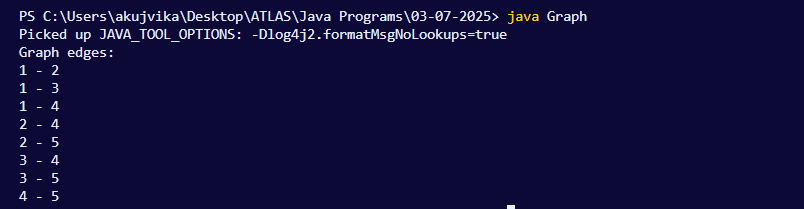
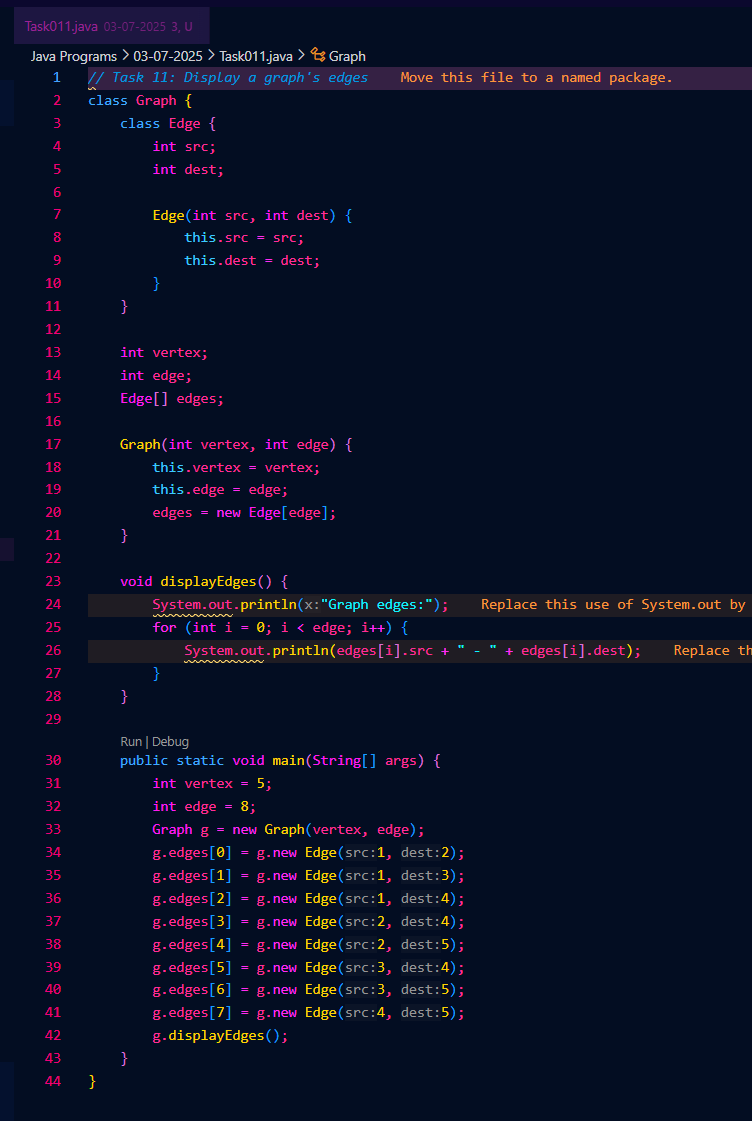
}

Int vertex;

Int edge;

}

Solution :



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

Home Task 👍

Q)Pre order and post order traversal.

A)



