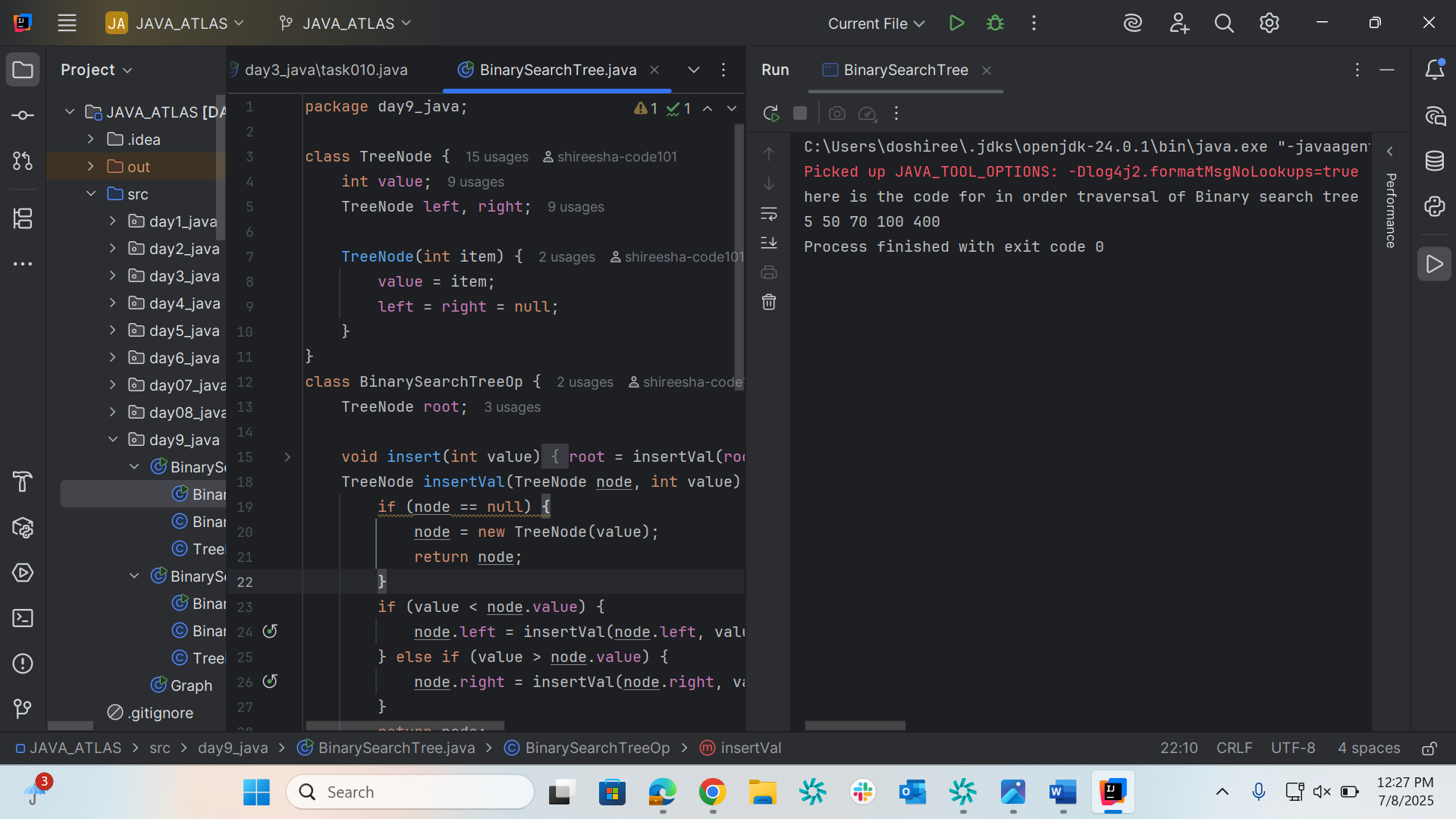
Day10\_java

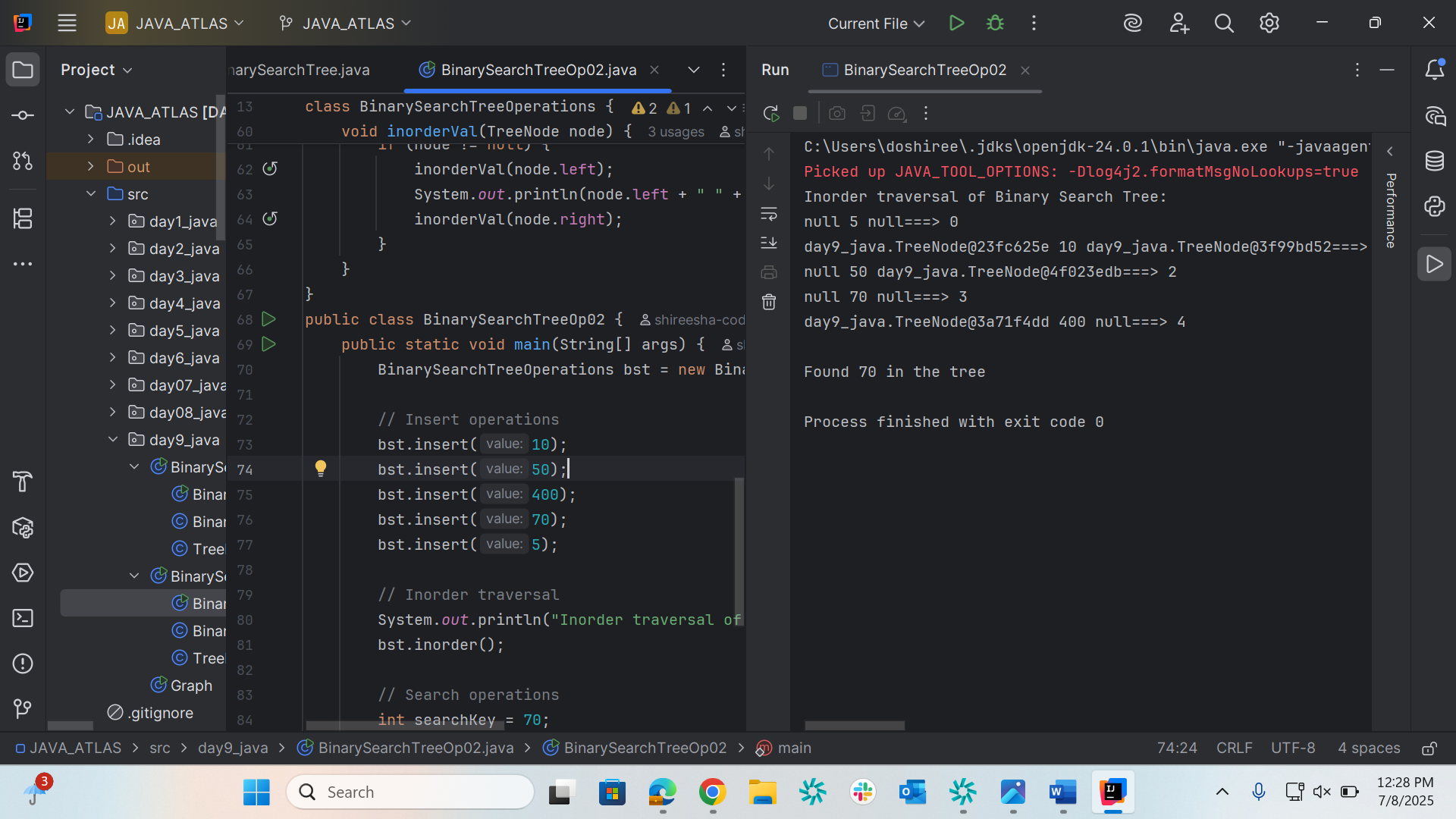
Task 1,2,3

package day9\_java;  
  
class TreeNode {  
 int value;  
 TreeNode left, right;  
  
 TreeNode(int item) {  
 value = item;  
 left = right = null;  
 }  
}  
class BinarySearchTreeOp {  
 TreeNode root;  
  
 void insert(int value) { // 10  
 root = insertVal(root, value); //root = null  
 }  
 TreeNode insertVal(TreeNode node, int value) { // null, 10 //  
 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(100);  
 bstobj.insert(50);  
 bstobj.insert(400);  
 bstobj.insert(70);  
 bstobj.insert(5);  
 System.*out*.println("here is the code for in order traversal of Binary search tree ");  
 bstobj.inorder();  
 }  
}



Task004

package day9\_java;  
  
class TreeNode1 {  
 int value;  
 TreeNode left, right;  
  
 TreeNode1(int value) {  
 this.value = value;  
 left = right = null;  
 }  
}  
  
class BinarySearchTreeOperations {  
 TreeNode root;  
 int i = 0;  
  
 // Constructor  
 public BinarySearchTreeOperations() {  
 this.root = null;  
 }  
  
 // Insert operations  
 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;  
 }  
  
 // Search operation  
 public TreeNode search(int key) {  
 TreeNode current = root;  
 while (current != null) {  
 if (key == current.value) {  
 return current;  
 } else if (key < current.value) {  
 current = current.left;  
 } else {  
 current = current.right;  
 }  
 }  
 return null;  
 }  
  
 // Traversal operations  
 void inorder() {  
 inorderVal(root);  
 }  
  
 void inorderVal(TreeNode node) {  
 if (node != null) {  
 inorderVal(node.left);  
 System.*out*.println(node.left + " " + node.value + " " + node.right + "===> " + i++);  
 inorderVal(node.right);  
 }  
 }  
}  
public class BinarySearchTreeOp02 {  
 public static void main(String[] args) {  
 BinarySearchTreeOperations bst = new BinarySearchTreeOperations();  
  
 // Insert operations  
 bst.insert(10);  
 bst.insert(50);  
 bst.insert(400);  
 bst.insert(70);  
 bst.insert(5);  
  
 // Inorder traversal  
 System.*out*.println("Inorder traversal of Binary Search Tree:");  
 bst.inorder();  
  
 // Search operations  
 int searchKey = 70;  
 TreeNode result = bst.search(searchKey);  
 if (result != null) {  
 System.*out*.println("\nFound " + searchKey + " in the tree");  
 } else {  
 System.*out*.println("\n" + searchKey + " not found in the tree");  
 }  
 }  
}



Graph   
package day9\_java;  
  
class Graph {  
  
 class Edge {  
 int src;  
 int dest;  
 }  
 int vertices;  
 int edges;  
 Edge[] edge;  
  
 Graph(int vertices, int edges) {  
 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 first = new Graph(noVertices, noEdges);  
  
 first.edge[0].src = 1;  
 first.edge[0].dest = 2;  
 first.edge[1].src = 1;  
 first.edge[1].dest = 3;  
 first.edge[2].src = 1;  
 first.edge[2].dest = 4;  
 first.edge[3].src = 2;  
 first.edge[3].dest = 4;  
 first.edge[4].src = 2;  
 first.edge[4].dest = 5;  
 first.edge[5].src = 3;  
 first.edge[5].dest = 4;  
 first.edge[6].src = 3;  
 first.edge[6].dest = 5;  
 first.edge[7].src = 4;  
 first.edge[7].dest = 5;  
  
 for(int i =0; i < noEdges; i++) {  
 System.*out*.println(first.edge[i].src+ " - " + first.edge[i].dest);  
 }  
 }  
}

