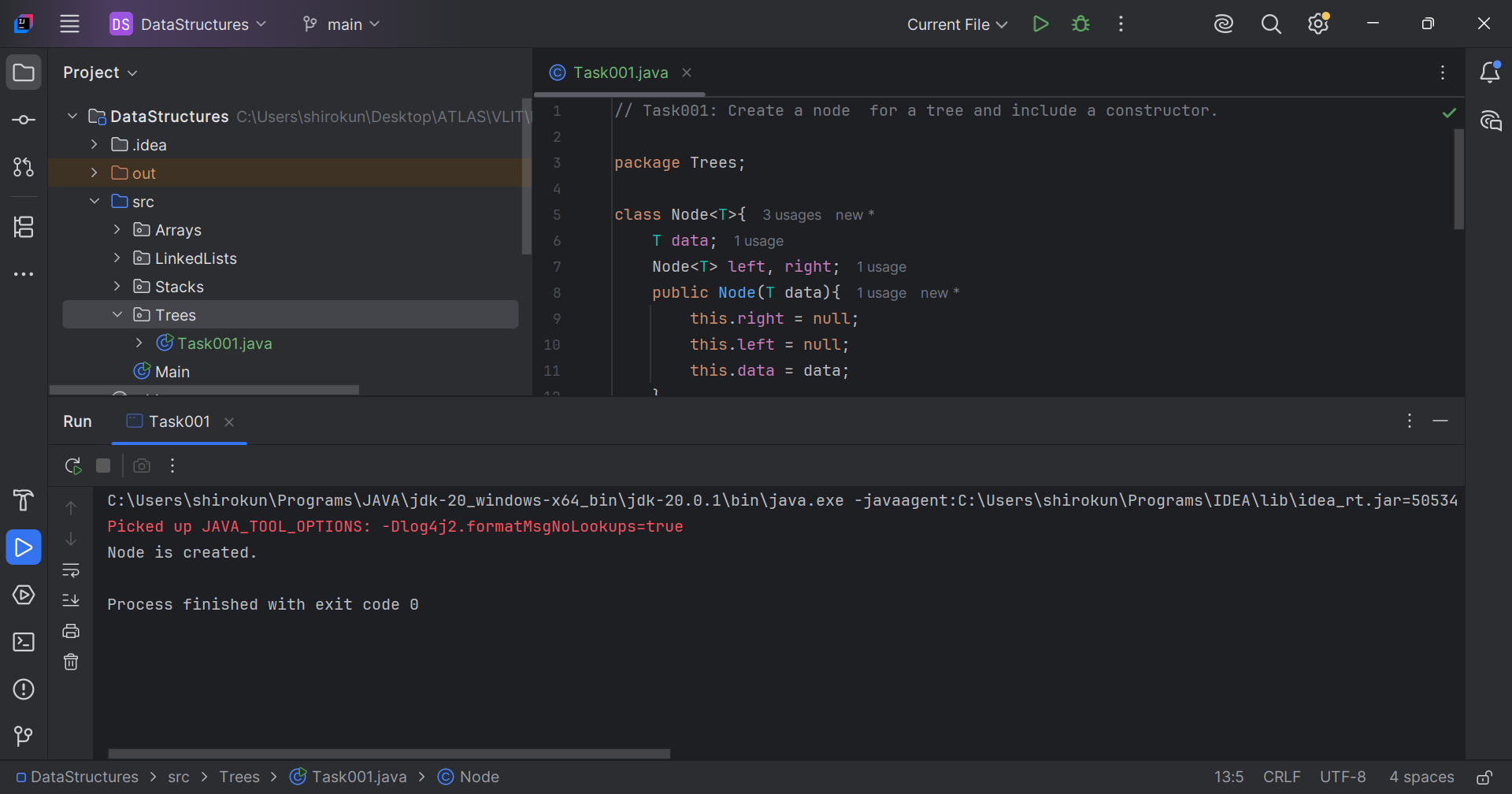
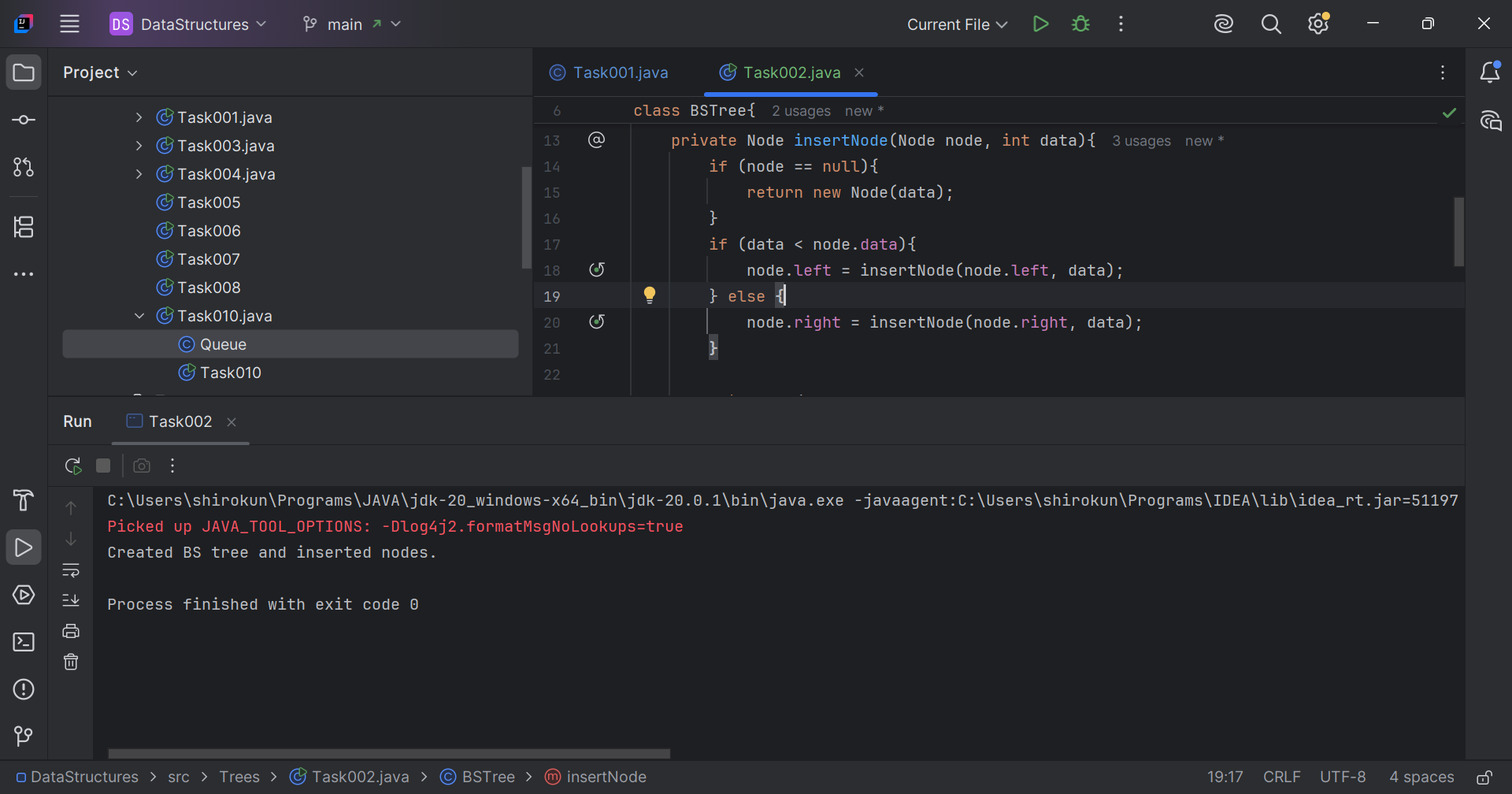
Day 15 – 05/07/2025

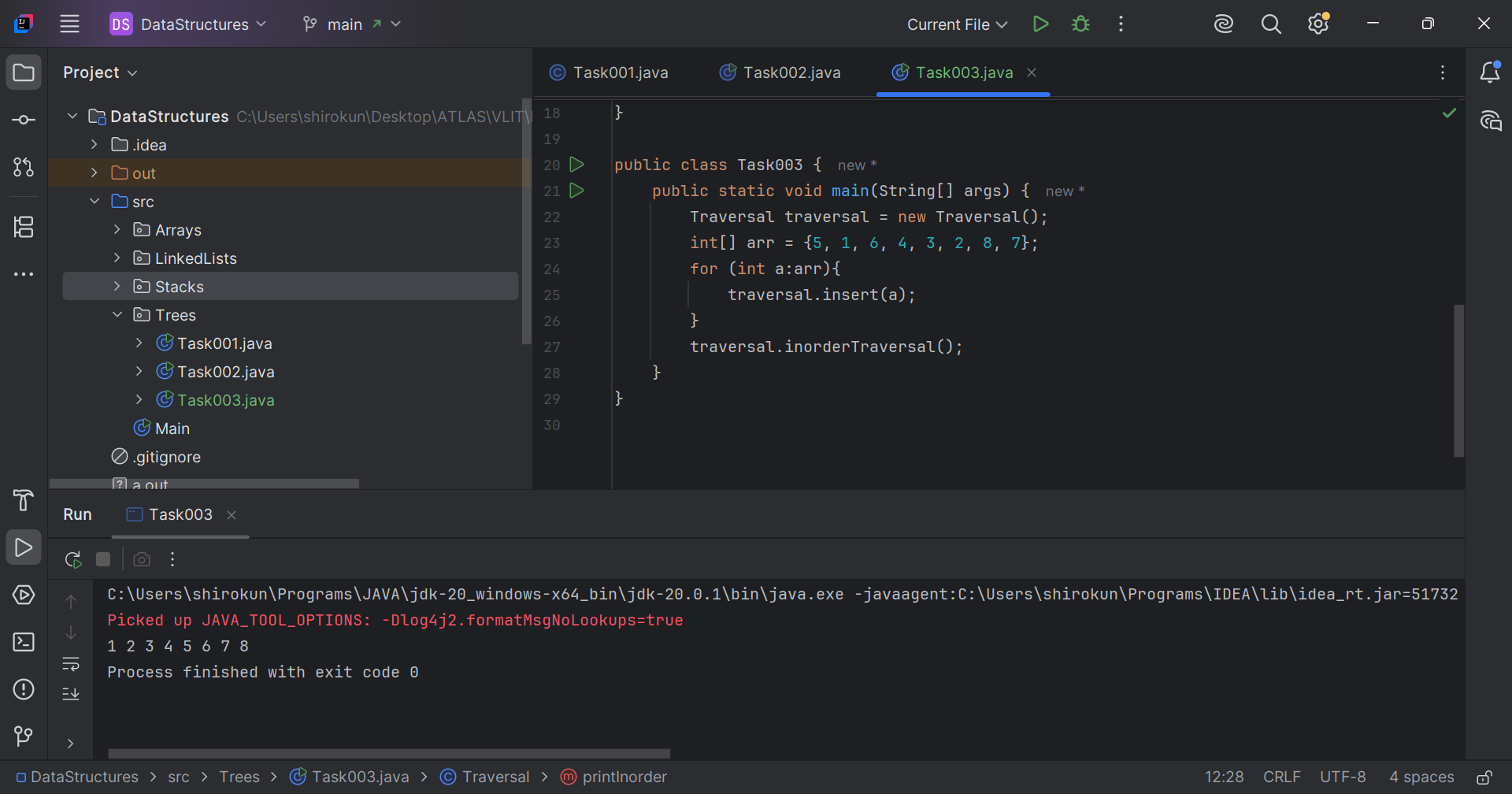
// Task001: Create a node for a tree and include a constructor.  
  
package Trees;  
  
class Node<T>{  
 T data;  
 Node<T> left, right;  
 public Node(T data){  
 this.right = null;  
 this.left = null;  
 this.data = data;  
 }  
  
 void display(){  
 System.*out*.println("Node is created.");  
 }  
}  
  
public class Task001 {  
 public static void main(String[] args) {  
 Node<String> node = new Node<>(" ");  
 node.display();  
 }  
}



// Task002: Create a class named Binary Search tree  
// in which you have 2 insert operations  
  
package Trees;  
  
class BSTree{  
 Node root;  
  
 public void insert(int data){  
 root = insertNode(root, data);  
 }  
  
 private Node insertNode(Node node, int data){  
 if (node == null){  
 return new Node(data);  
 }  
 if (data < node.data){  
 node.left = insertNode(node.left, data);  
 } else {  
 node.right = insertNode(node.right, data);  
 }  
 return node;  
 }  
  
}  
  
public class Task002 {  
 public static void main(String[] args) {  
 BSTree bsTree = new BSTree();  
 int[] arr = {5, 1, 6, 4, 3, 2, 8, 7};  
 for (int a:arr){ bsTree.insert(a);}  
 System.*out*.println("Created BS tree and inserted nodes.");  
 }  
}



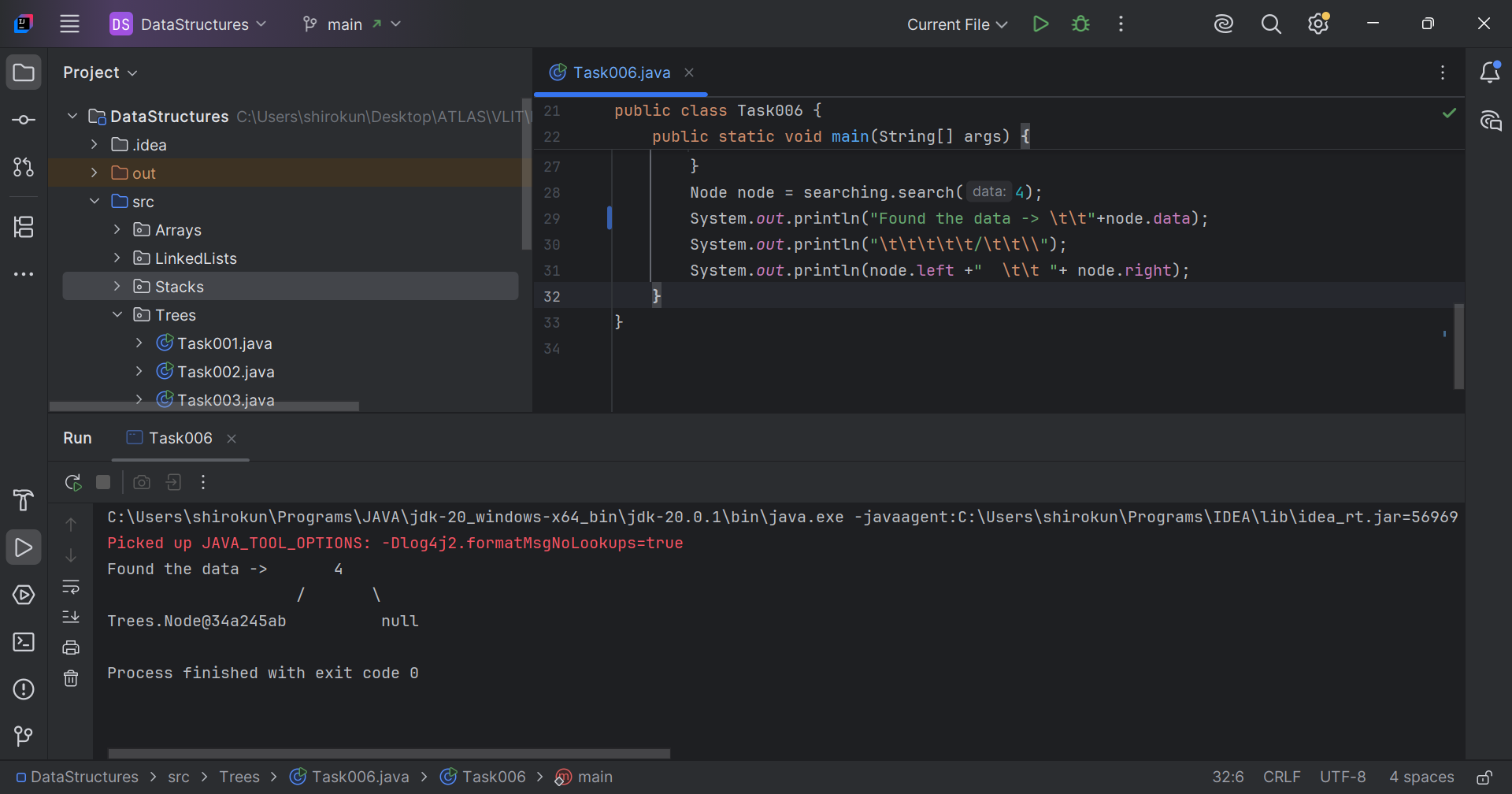
// Task003: Inorder traversal.  
  
package Trees;  
  
class Traversal extends BSTree {  
  
 void inorderTraversal() {  
 printInorder(root);  
 }  
  
 private void printInorder(Node node) {  
 if (node != null) {  
 printInorder(node.left);  
 System.*out*.print(node.data + " ");  
 printInorder(node.right);  
 }  
 }  
}  
  
public class Task003 {  
 public static void main(String[] args) {  
 Traversal traversal = new Traversal();  
 int[] arr = {5, 1, 6, 4, 3, 2, 8, 7};  
 for (int a:arr){  
 traversal.insert(a);  
 }  
 traversal.inorderTraversal();  
 }  
}



Q5. What are applications of trees?  
Ans. Applications of trees:

* File Systems: Trees are used to represent the directory structure of file systems, where each node represents a directory or file.
* Database Indexing: Trees are used in database indexing to efficiently retrieve data. B-trees and B+ trees are commonly used in databases.
* Compilers: Trees are used in compilers to represent the parse trees of source code, which are used to analyze the syntax and semantics of the code.
* Decision Trees: Decision trees are used in machine learning and data analysis to classify data and make predictions based on a set of features.
* Webpage Navigation: Trees are used in webpage navigation to represent the structure of a website and facilitate navigation.
* Network Topology: Trees are used to represent network topology, where each node represents a device or router.

// Task006: Binary searching.  
  
package Trees;  
  
class Searching extends Traversal{  
 public Node search(int data){  
 Node current = root;  
 while (current != null){  
 if (current.data == data){  
 return current;  
 } else if (current.data < data) {  
 current = current.right;  
 } else {  
 current = current.left;  
 }  
 }  
 return null;  
 }  
}  
  
public class Task006 {  
 public static void main(String[] args) {  
 Searching searching = new Searching();  
 int[] arr = {5, 1, 6, 4, 3, 2, 8, 7};  
 for (int a:arr){  
 searching.insert(a);  
 }  
 Node node = searching.search(4);  
 System.*out*.println("Found the data -> \t\t"+node.data);  
 System.*out*.println("\t\t\t\t\t/\t\t\\");  
 System.*out*.println(node.left +" \t\t "+ node.right);  
 }  
}



Q7. What are different types of binary trees?  
Ans. Types of binary trees:

* Full Binary Tree: A full binary tree is a binary tree where every node has either 0 or 2 children.
* Complete Binary Tree: A complete binary tree is a binary tree where every level is fully filled, except for possibly the last level, which is filled from left to right.
* Perfect Binary Tree: A perfect binary tree is a binary tree where every level is fully filled, and all leaves are at the same level.
* Balanced Binary Tree: A balanced binary tree is a binary tree where the height of the left and right subtrees of every node differs by at most 1.
* Skewed Binary Tree: A skewed binary tree is a binary tree where most nodes have only one child, resulting in an unbalanced tree.

Q8. What are applications of graphs?  
Ans. Applications of graphs –

* Traffic and Transportation: Graphs are used to model traffic patterns and optimize routes for transportation networks.
* Logistics and Supply Chain Management: Graphs are used to optimize routes and schedules for delivery and supply chain management.
* Biological Networks: Graphs are used to represent biological networks, including protein-protein interaction networks and gene regulatory networks.
* Scheduling: Graphs are used in scheduling algorithms to optimize resource allocation and minimize conflicts.

Q9. What are different types of graphs?  
Ans. Types of graphs:

* Directed Graph: Edges have direction, and the graph may not be symmetric.
* Undirected Graph: Edges do not have direction, and the graph is symmetric.
* Weighted Graph: Edges have weights or labels associated with them.
* Unweighted Graph: Edges do not have weights or labels.

// Task010: Display graph edges.  
  
package Trees;  
  
class Graph{  
 static class Edge{  
 int src, destination;  
  
 public Edge(int src, int destination){  
 this.src = src;  
 this.destination = destination;  
 }  
 }  
 int vertices, edges;  
 Edge[] edge;  
  
 public 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(0, 0);  
 }  
 }  
  
 public void addEdge(int index, int src, int destination){  
 if (index >= 0 && index < edges){  
 edge[index].src = src;  
 edge[index].destination = destination;  
 }  
 }  
  
 public void display(){  
 for (int i = 0; i < edges; i++) {  
 System.*out*.println("Edge "+(i+1)+": "+  
 edge[i].src+" -> "+edge[i].destination);  
 }  
 }  
}

public class Task010 {  
 public static void main(String[] args) {  
 int vertices = 5, edges = 7;  
 Graph graph = new Graph(vertices, edges);  
 graph.addEdge(0, 1, 2);  
 graph.addEdge(1, 1, 3);  
 graph.addEdge(2, 1, 4);  
 graph.addEdge(3, 2, 4);  
 graph.addEdge(4, 2, 5);  
 graph.addEdge(5, 3, 4);  
 graph.addEdge(6, 3, 5);  
 graph.display();  
 }  
}

