Day 15

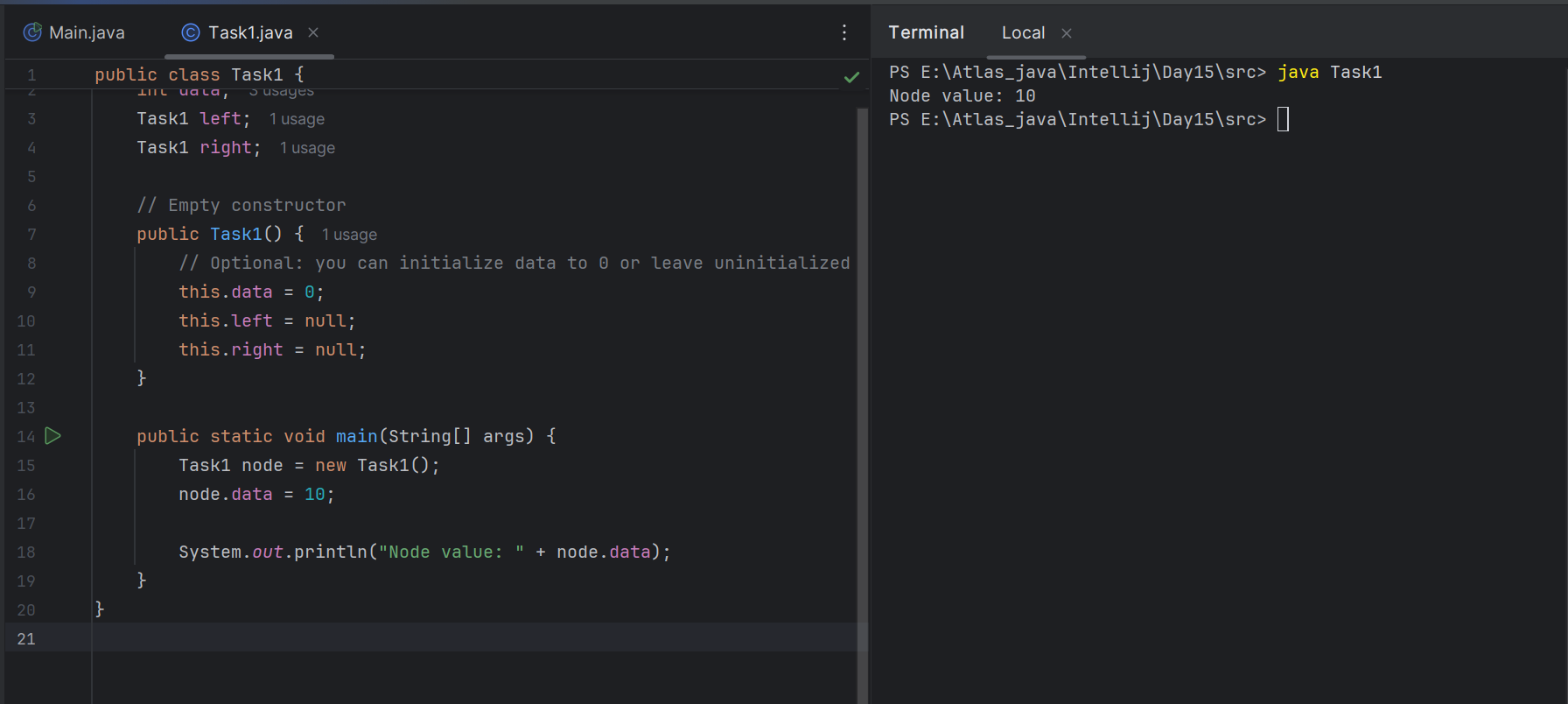
Employee ID: 201933938

Login ID: iamasif

Name: Shaik Asif

**Task1:**

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



4 min

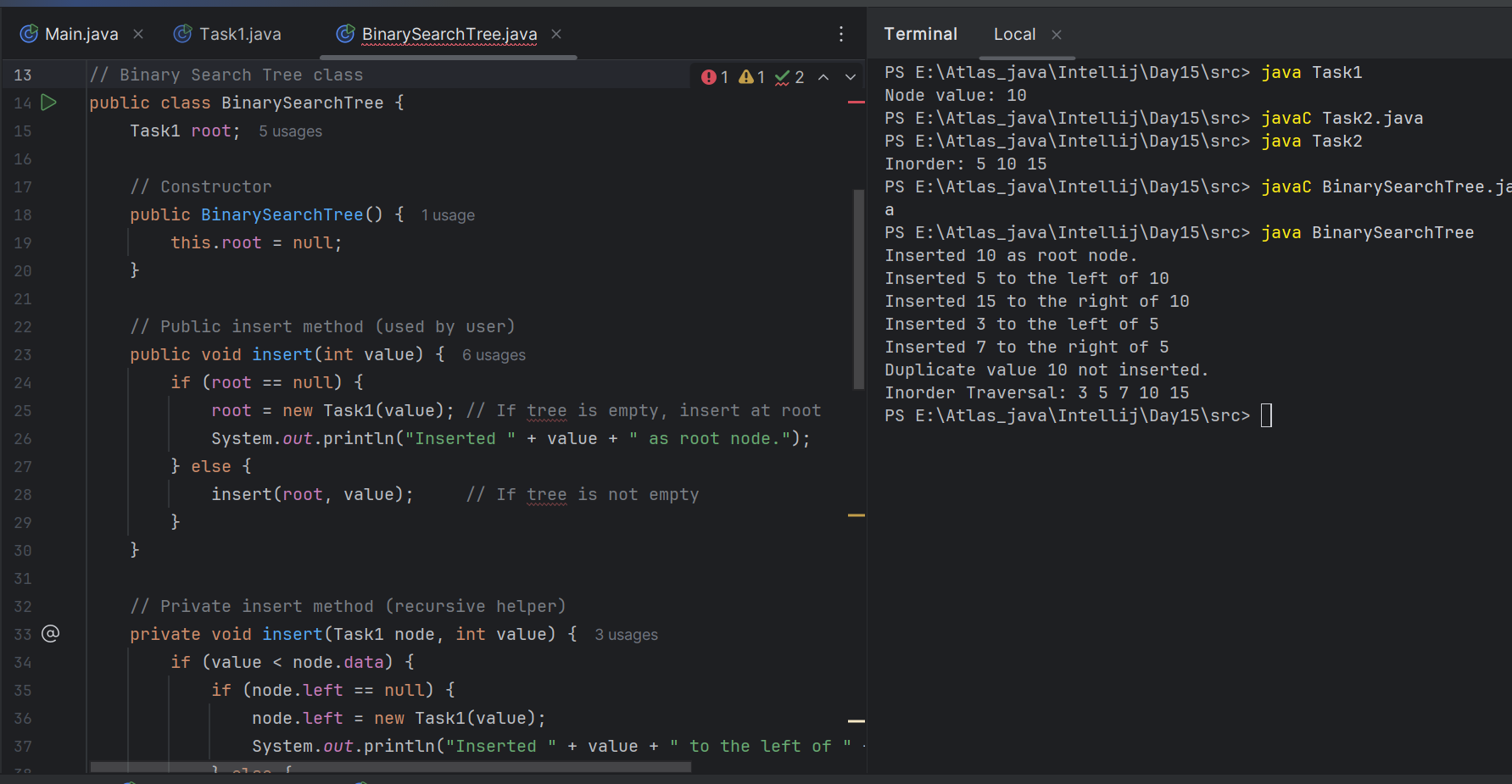
10.54 to 10.58

**Task 2:**

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

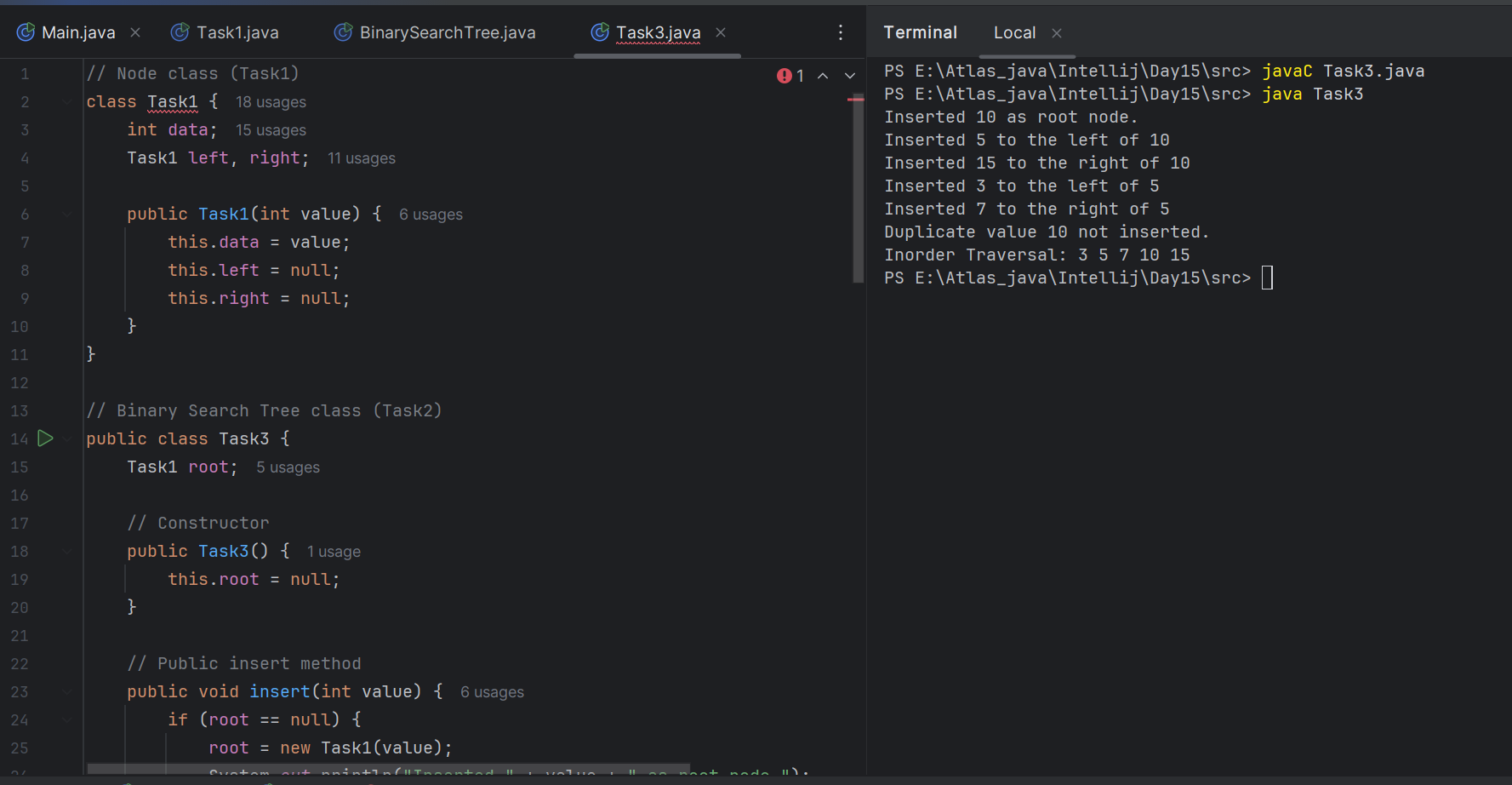
1 insert —----> for inserting if the tree is empty

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



**Task 3:**

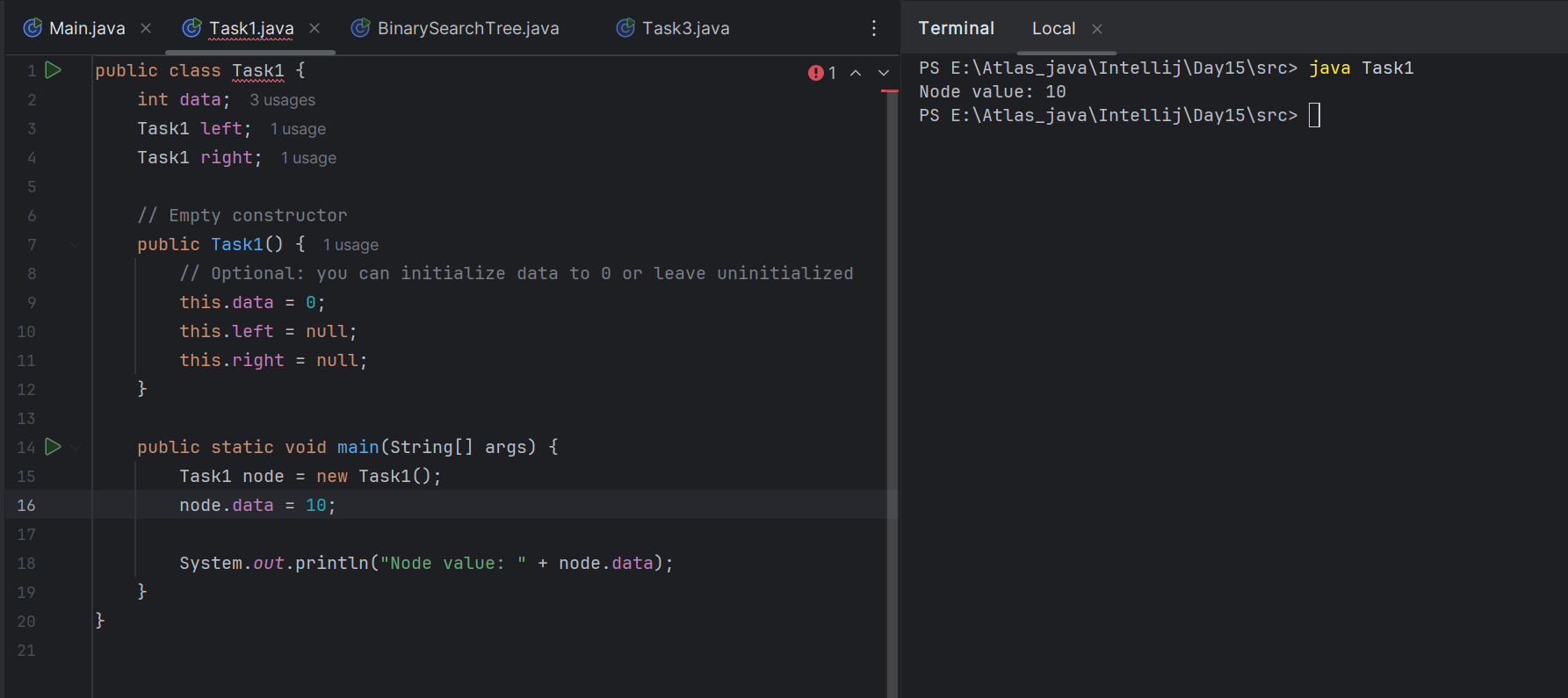
Ionorder travel of the above code snippets from task 1 and Task 2



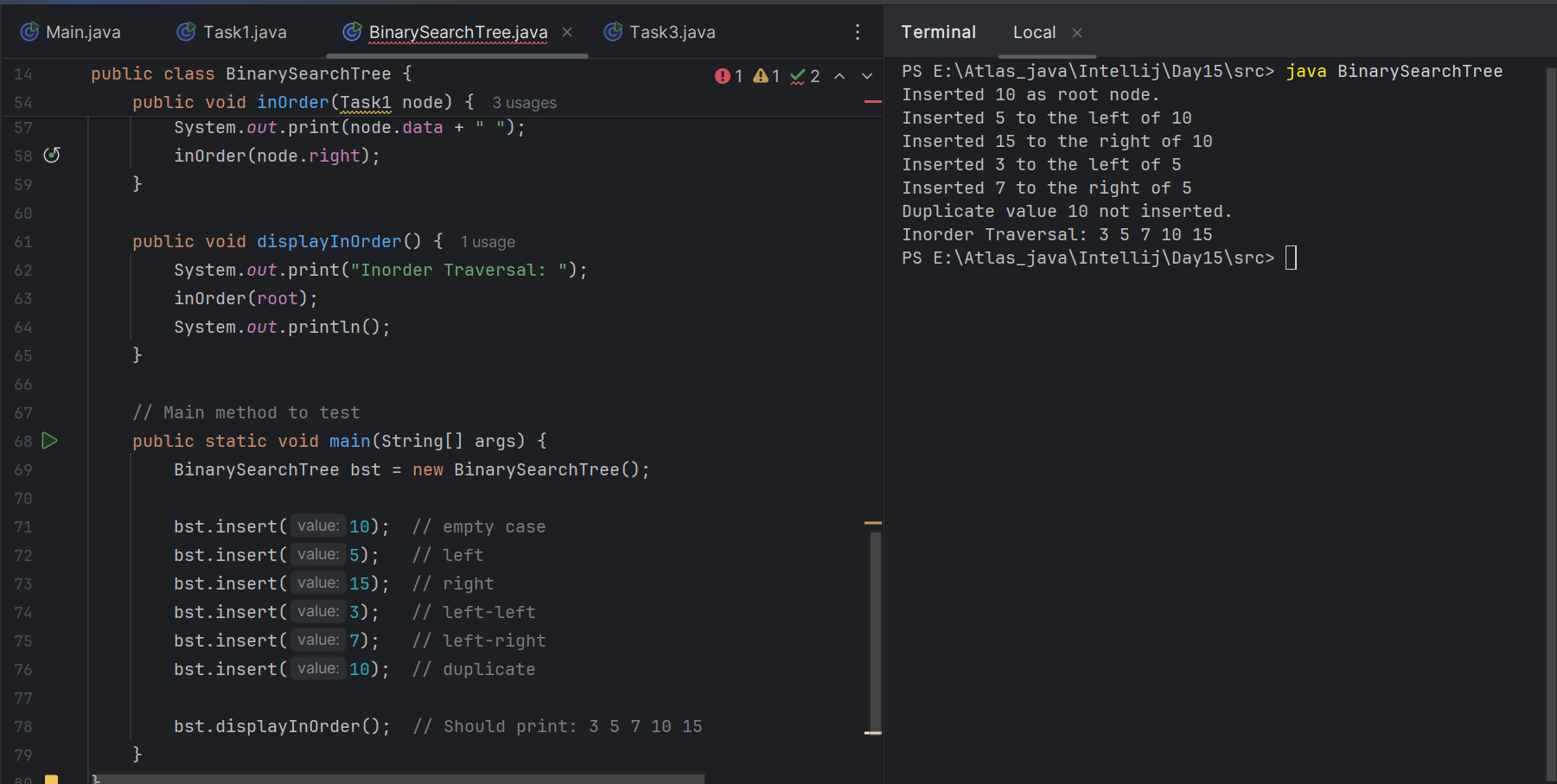
**Task 4:**

Create a main method Task 1, 2 and 3

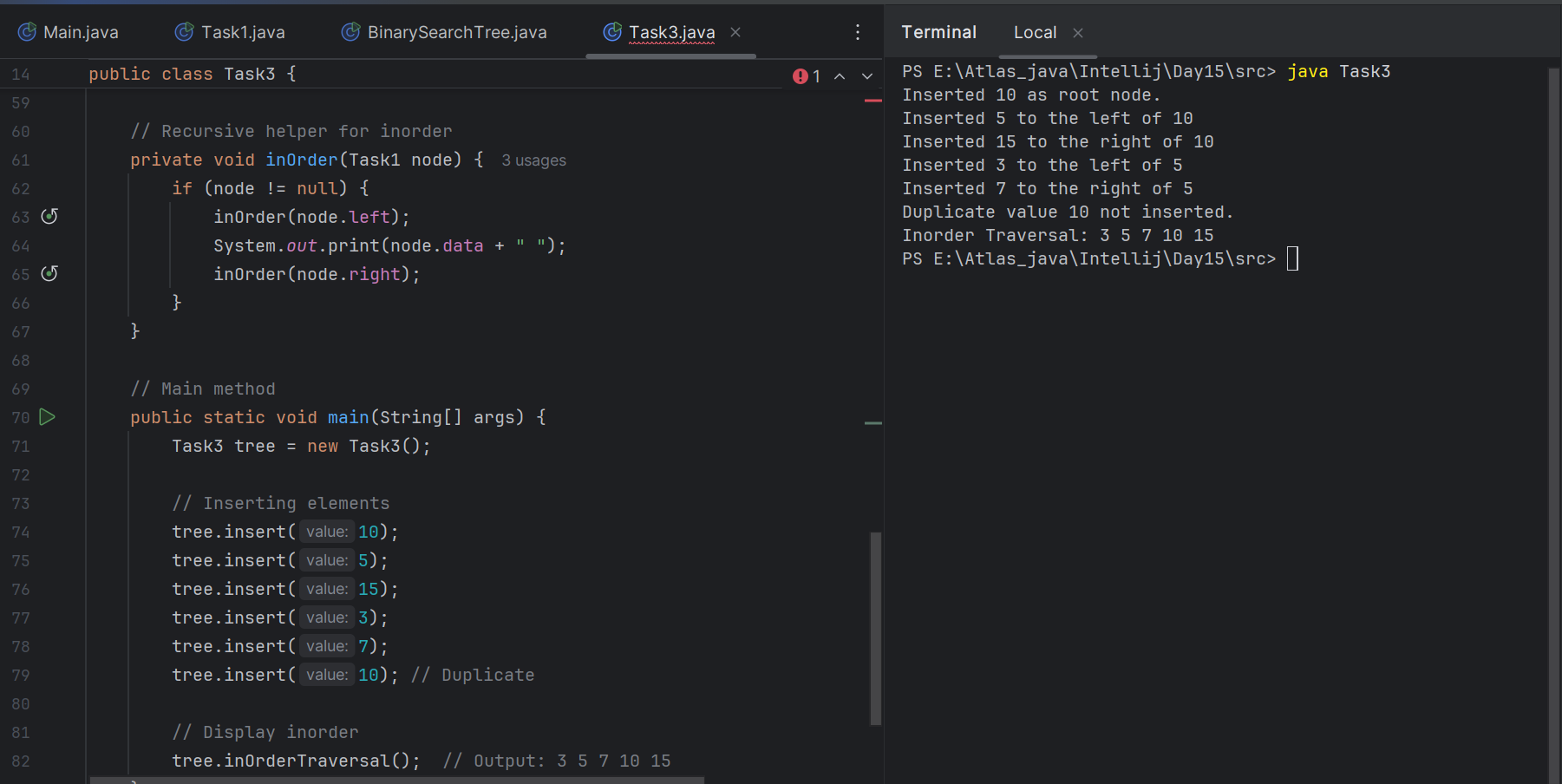
**Task1 main:**



**Task2 main:**



**Task3 main:**



And run the code..

12.04 to 12.09

Solution for Task 1 to task 4

In order Binary Search tree..

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

**Done**

#### 1. Hierarchical Data Representation

* Trees naturally represent hierarchical structures.
* Example: File systems (folders inside folders), organization charts.

#### 2. Binary Search Tree (BST)

* Used to store data in sorted order for quick searching, insertion, and deletion.
* Time complexity: O(log n) for balanced trees.

#### 3. Heaps (Priority Queues)

* Binary Heaps (Min-Heap / Max-Heap) are used in:  
  + Heap Sort
  + Priority Queues
  + Task Scheduling

#### 4. Trie (Prefix Tree)

* Special tree used for fast word lookups.
* Common in:  
  + Auto-complete
  + Spell checkers
  + IP routing

#### 5. Expression Trees

* Used to evaluate mathematical expressions.
* Example: Parsing ((2 + 3) \* (4 - 1))

#### 6. Routing and Decision Trees

* Used in:  
  + Network routing algorithms
  + AI decision-making
  + Game trees in AI (like in chess)

#### 7. XML / JSON Parsing

* Trees represent the nested structure of data in markup languages.
* Used in web development and data transfer.

#### 8. Databases

* B-Trees and B+ Trees are used in:  
  + Indexing
  + Efficient searching in large databases (e.g., MySQL, PostgreSQL)

#### 9. Compiler Design

* Abstract Syntax Trees (AST) are used to parse and analyze source code.

#### 10. Machine Learning

* Decision Trees, Random Forests, and Gradient Boosted Trees are key algorithms in predictive modeling.

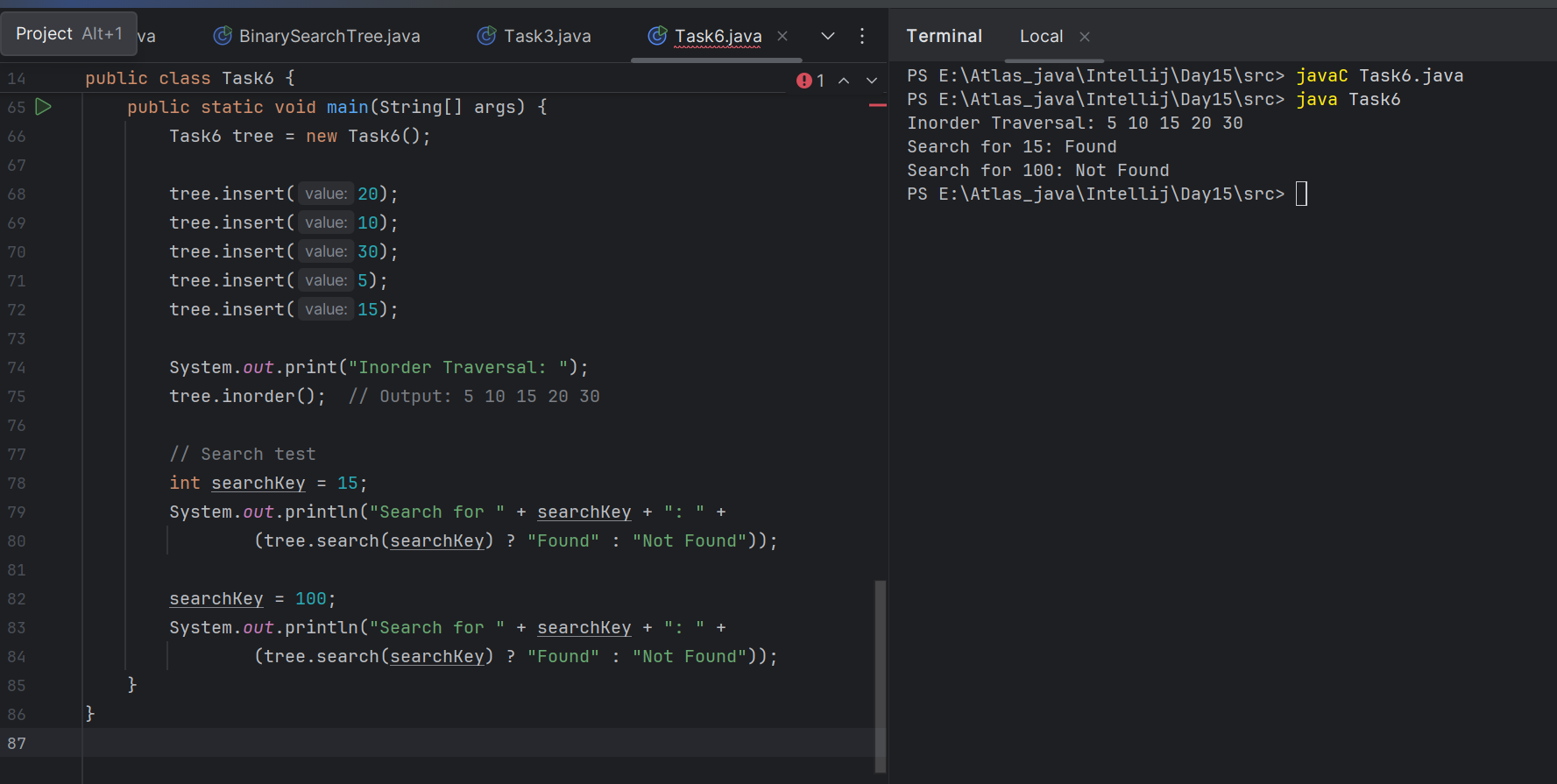
**Task 6:**

Create a binary search operation on tree

Hint:

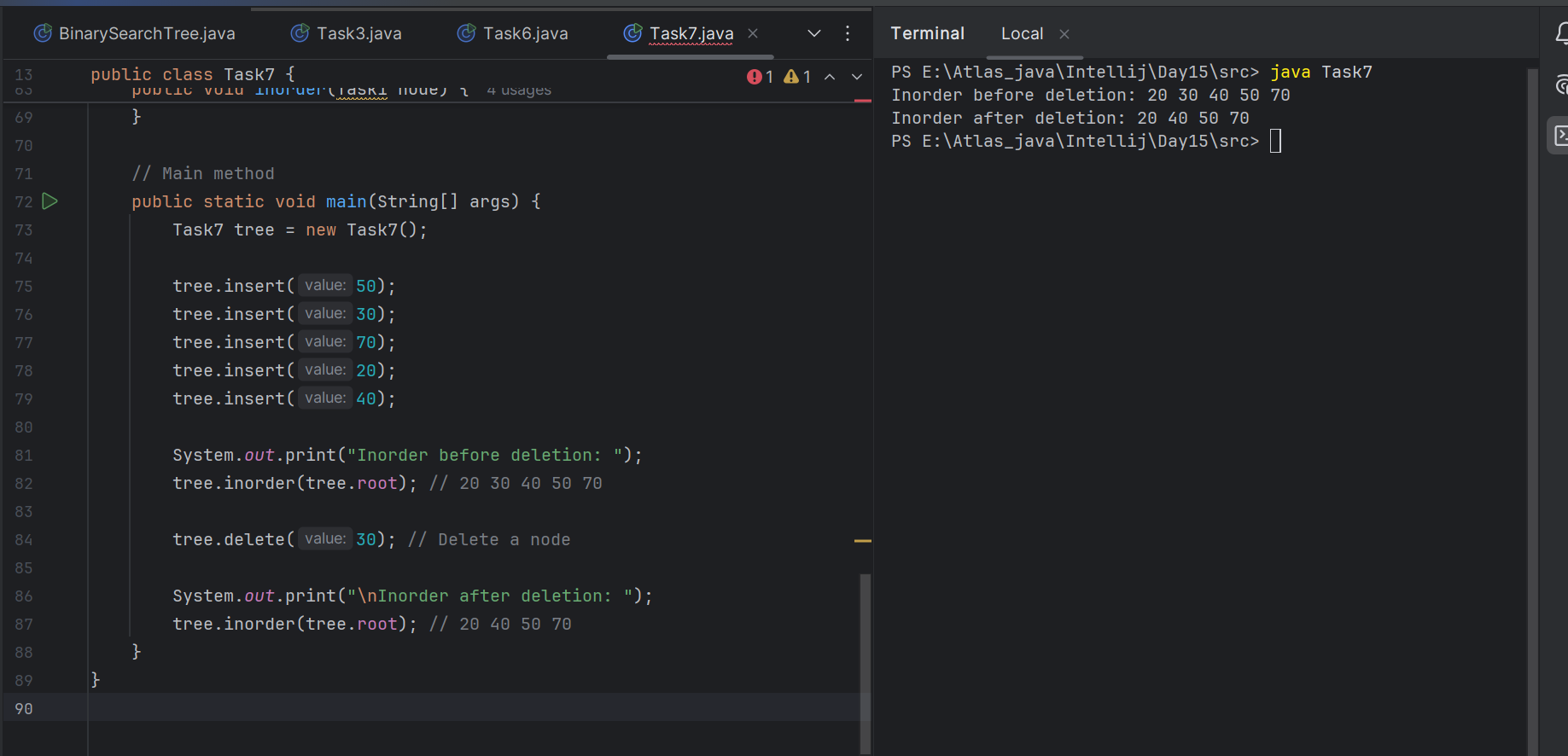
Create a node

Class for binary search



**Task7:**

wap for delete operation

****

Task8: **Types of Binary tree.**

**Rooted Binary Tree**

• A rooted binary tree is a binary tree that

satisfies the following 2 properties:

– It has a root node.

– Each node has at most 2 children.

**Full/Strictly Binary Tree**

• A binary tree in which every node has either 0 or 2

children is called as a Full binary tree.

• Full binary tree is also called as Strictly binary tree.

**Complete /Perfect Binary Tree**

• A complete binary tree is a binary tree that satisfies

the following 2 properties:

– Every internal node has exactly 2 children.

– All the leaf nodes are at the same level.

**Almost Complete Binary Tree**

• An almost complete binary tree is a binary tree that satisfies

the following 2 properties-

– All the levels are completely filled except possibly the last level.

– The last level must be strictly filled from left to right.

**Skewed Binary Tree**

• A skewed binary tree is a binary tree that satisfies the following 2 properties-

• All the nodes except one node has one and only one child.

• The remaining node has no child.

OR

• A skewed binary tree is a binary tree of n nodes such that its depth is (n-1).

**Differences Between Types of Binary Trees**

| **Type of Binary Tree** | **Definition / Rule** | **Example Structure** | **Special Notes** |
| --- | --- | --- | --- |
| **Rooted Binary Tree** | A binary tree with a single designated root node | Starts from a root and connects down to children | Basic form of all binary trees |
| **Full / Strictly Binary** | Every node has 0 or 2 children only | All internal nodes have 2 children; leaves have none | No node has only one child |
| **Complete Binary Tree** | All levels are completely filled, except the last level is filled left to right | Top-down filled left-to-right | Used in heaps |
| **Perfect Binary Tree** | A special complete tree where all internal nodes have 2 children and all leaves are at the same level | Fully symmetrical tree | A perfect triangle shape |
| **Almost Complete Binary** | Similar to complete, but last level may have gaps (not filled left to right) | One or two nodes missing in the last level | Nearly complete, not strict |
| **Skewed Binary Tree** | All nodes have only one child (either all left or all right) | Linear — like a linked list | Worst case for BST; height = number of nodes |

Task 9:

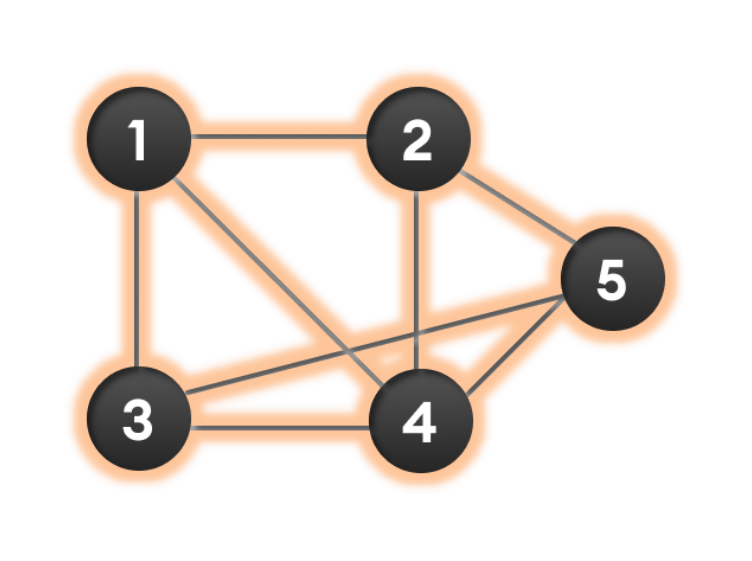
Applications of Graphs:

| **Application Area** | **Graph Role** |
| --- | --- |
| Internet | Routing, Web crawling |
| Maps & GPS | Shortest path, navigation |
| Social Media | Friend/follow connections |
| AI | Game trees, search spaces |
| Project Management | Task dependencies, scheduling |
| Biology | Gene and protein networks |
| Recommendations | User-item relationships |
| Cybersecurity | Attack path tracing, anomaly detection |

Task 10:

Types of Graphs:

| **Type** | **Key Feature** | **Example Use Case** |
| --- | --- | --- |
| Directed | Edges have direction | Web links |
| Undirected | Edges have no direction | Friendships |
| Weighted | Edges have costs | Maps, networks |
| Unweighted | All edges equal | Social graphs |
| Cyclic | Has loops | Electrical circuits |
| Acyclic (DAG) | No loops | Task scheduling |
| Connected | All nodes reachable | Network design |
| Disconnected | Some nodes isolated | Multiple subnetworks |
| Complete | Every node connected to every other | Fully connected systems |
| Bipartite | Two sets, no same-set connections | Job assignments |
| Regular | Same number of connections per node | Peer-to-peer networks |
| Tree | Acyclic, connected, hierarchical | File systems, decision trees |



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{

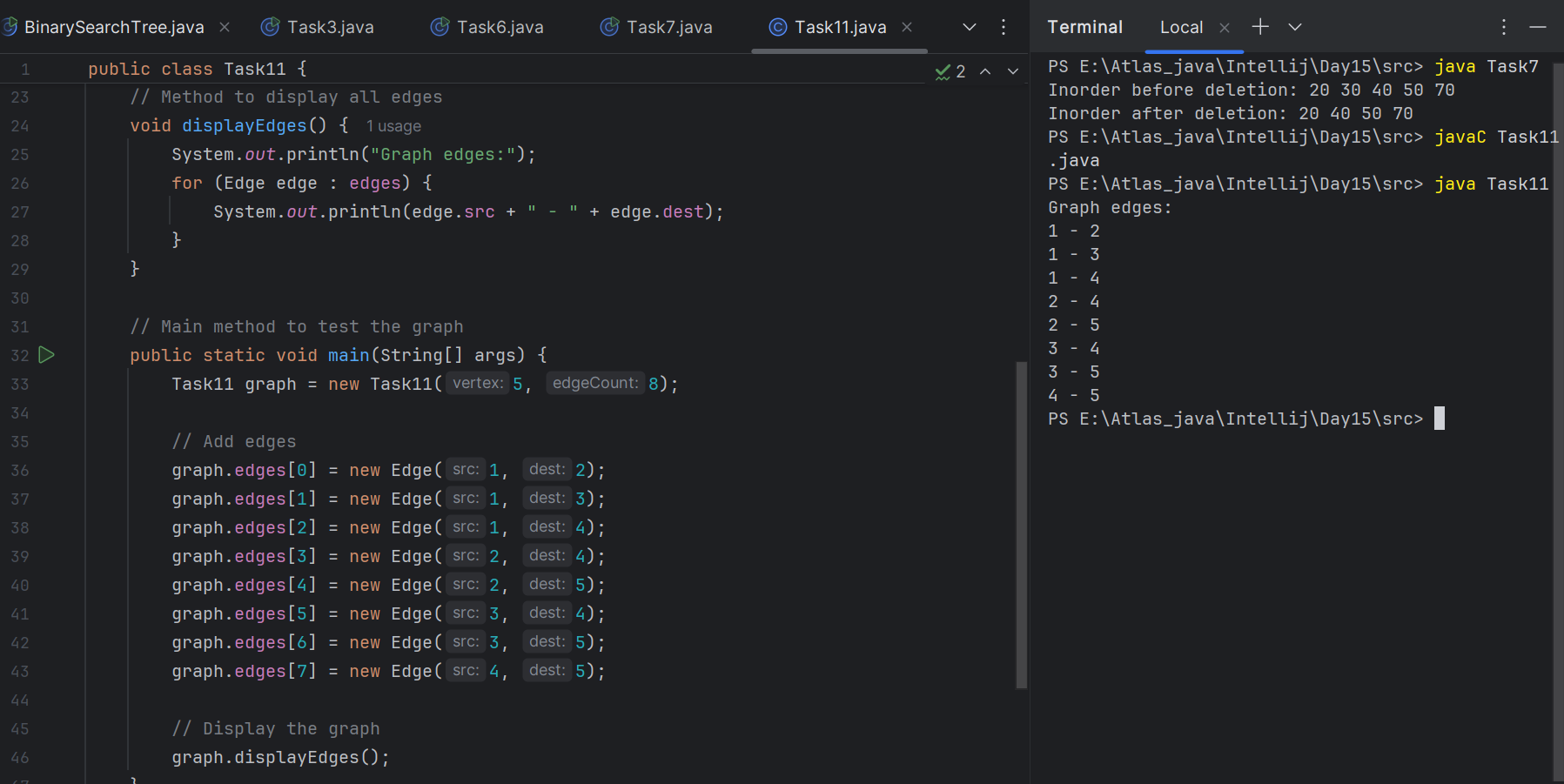
Int start/src;

Int end/dest;

}

Int vertex;

}



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

Graphs

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

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**info box**

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

<https://drive.google.com/drive/folders/1LwhNov1s1-vHzF9GPAObLSnP9kAvipmw?usp=drive_link>

Binary tree delete op ..along with insertion and traversal inorder …

<https://www.codecademy.com/learn/graph-data-structures-java/modules/graphs-java/cheatsheet>

Graphs in data structures.. Along with code – juz for reference..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*