Task 1:

Create a node  for a tree and include a constructor.

public class task01 {

    static class Node {

        int data;

        Node left;

        Node right;

        Node(int data) {

            this.data = data;

            left = null;

            right = null;

        }

    }

    public static void main(String[] args) {

        Node root = new Node(10);

        Node node1 = new Node(20);

        Node node2 = new Node(30);

        root.left = node1;

        root.right = node2;

        System.out.println("Root node: " + root.data);

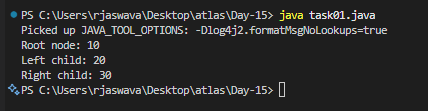
        System.out.println("Left child: " + root.left.data);

        System.out.println("Right child: " + root.right.data);

    }

}

Output:



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 has 1 or more nodes

public class task02 {

    static class Node {

        int data;

        Node left, right;

        Node(int data) {

            this.data = data;

            left = right = null;

        }

    }

    static class BinarySearchTree {

        Node root;

        BinarySearchTree() {

            root = null;

        }

        void insertFirst(int data) {

            if(root == null) {

                root = new Node(data);

                System.out.println("Inserted " + data + " as root");

            }

        }

        void insert(int data) {

            if(root == null) {

                insertFirst(data);

                return;

            }

            Node current = root;

            Node parent = null;

            while(current != null) {

                parent = current;

                if(data < current.data) {

                    current = current.left;

                } else {

                    current = current.right;

                }

            }

            if(data < parent.data) {

                parent.left = new Node(data);

                System.out.println("Inserted " + data + " to left");

            } else {

                parent.right = new Node(data);

                System.out.println("Inserted " + data + " to right");

            }

        }

    }

    public static void main(String[] args) {

        BinarySearchTree tree = new BinarySearchTree();

        tree.insertFirst(50);

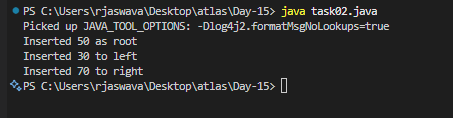
        tree.insert(30);

        tree.insert(70);

    }

}

Output:



Task 3:

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

public class task03 {

    static class Node {

        int data;

        Node left, right;

        Node(int data) {

            this.data = data;

            left = right = null;

        }

    }

    static class BinarySearchTree {

        Node root;

        BinarySearchTree() {

            root = null;

        }

        void insert(int data) {

            if(root == null) {

                root = new Node(data);

                return;

            }

            Node current = root;

            Node parent = null;

            while(current != null) {

                parent = current;

                if(data < current.data)

                    current = current.left;

                else

                    current = current.right;

            }

            if(data < parent.data)

                parent.left = new Node(data);

            else

                parent.right = new Node(data);

        }

        void inorder(Node node) {

            if(node != null) {

                inorder(node.left);

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

                inorder(node.right);

            }

        }

        void printInorder() {

            System.out.print("Inorder traversal: ");

            inorder(root);

            System.out.println();

        }

    }

    public static void main(String[] args) {

        BinarySearchTree tree = new BinarySearchTree();

        tree.insert(50);

        tree.insert(30);

        tree.insert(70);

        tree.insert(20);

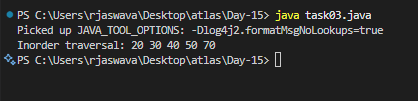
        tree.insert(40);

        tree.printInorder();

    }

}

Output:



Task 4:

Create  a main method Task 1, 2 and 3

And run the code.

public class task04 {

    static class Node {

        int data;

        Node left, right;

        Node(int data) {

            this.data = data;

            left = right = null;

        }

    }

    static class BinarySearchTree {

        Node root;

        BinarySearchTree() {

            root = null;

        }

        void insertFirst(int data) {

            if(root == null) {

                root = new Node(data);

                System.out.println("Inserted root: " + data);

            }

        }

        void insert(int data) {

            if(root == null) {

                insertFirst(data);

                return;

            }

            Node current = root;

            Node parent = null;

            while(current != null) {

                parent = current;

                if(data < current.data)

                    current = current.left;

                else

                    current = current.right;

            }

            if(data < parent.data) {

                parent.left = new Node(data);

                System.out.println("Inserted left: " + data);

            } else {

                parent.right = new Node(data);

                System.out.println("Inserted right: " + data);

            }

        }

        void inorder(Node node) {

            if(node != null) {

                inorder(node.left);

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

                inorder(node.right);

            }

        }

    }

    public static void main(String[] args) {

        BinarySearchTree tree = new BinarySearchTree();

        System.out.println("Creating and inserting nodes...");

        tree.insertFirst(50);

        tree.insert(30);

        tree.insert(70);

        tree.insert(20);

        tree.insert(40);

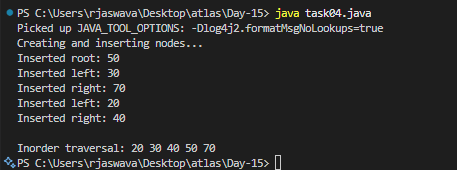
        System.out.print("\nInorder traversal: ");

        tree.inorder(tree.root);

    }

}

Output:



Task 5:

Applications of Trees

* HTML DOM ( Document Object Model)
* Database Indexing
* File Systems
* Network Routing

Task 6:

 Create  a binary search operation on tree

Hint:

Create a node

Class for binary search

public class task06 {

    static class Node {

        int data;

        Node left, right;

        Node(int data) {

            this.data = data;

            left = right = null;

        }

    }

    static class BinarySearchTree {

        Node root;

        BinarySearchTree() {

            root = null;

        }

        void insert(int data) {

            if(root == null) {

                root = new Node(data);

                System.out.println("Added: " + data);

                return;

            }

            Node current = root;

            while(true) {

                if(data < current.data) {

                    if(current.left == null) {

                        current.left = new Node(data);

                        System.out.println("Added: " + data);

                        break;

                    }

                    current = current.left;

                } else {

                    if(current.right == null) {

                        current.right = new Node(data);

                        System.out.println("Added: " + data);

                        break;

                    }

                    current = current.right;

                }

            }

        }

        void search(int data) {

            Node current = root;

            while(current != null) {

                if(current.data == data) {

                    System.out.println("Found: " + data);

                    return;

                }

                if(data < current.data) {

                    current = current.left;

                } else {

                    current = current.right;

                }

            }

            System.out.println("Not Found: " + data);

        }

    }

    public static void main(String[] args) {

        BinarySearchTree tree = new BinarySearchTree();

        tree.insert(50);

        tree.insert(30);

        tree.insert(70);

        tree.insert(20);

        tree.insert(40);

        System.out.println("\nSearching for elements:");

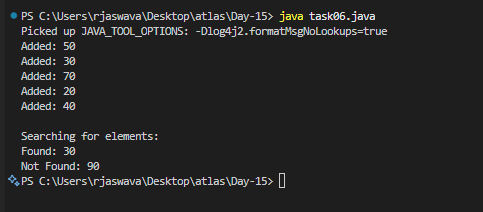
        tree.search(30);

        tree.search(90);

    }

}

Output:



Task 7:

Types of binary trees

* Full Binary Tree
* Every node has 0 or 2 children
* No node has exactly 1 child
* Complete Binary Tree
* All levels are filled except possibly the last level
* Last level has nodes as left possible
* Perfect Binary Tree
* All internal nodes have 2 children
* All leaves are the same level
* Balanced Binary Tree
* Height difference between left and right subtrees is at most 1
* For every node, left and right subtrees heights differ by at most 1
* Degenerate Binary Tree
* Every parent has exactly one child
* Like a linked list
* Binary Search Tree
* Left subtree has smaller values
* Right subtree has larger values
* No duplicate values

Task 8:

Applications of Graphs

* GPS and Navigation
* Road networks
* Google maps
* Navigation systems
* Computer Networks
* Internet routing
* Network topology
* Network design
* Transportation Systems
* Flight routes
* Railway networks
* Public transit systems
* Shipping routes
* Web Applications
* Web page links
* Website structure
* Search engine crawling

Task 9:

Types of Graphs:

Types of Graphs in DSA:

1. Basic Graph Types:

a) Undirected Graph

- No direction on edges

- Example: A --- B

b) Directed Graph (Digraph)

- Has direction on edges

- Example: A --→ B

2. Based on Edge Properties:

a) Weighted Graph

- Edges have values

b) Unweighted Graph

- No values on edges

3. Based on Connectivity:

a) Connected Graph

- All vertices have path to others

b) Disconnected Graph

- Some vertices not reachable

4. Based on Cycles:

a) Cyclic Graph

- Contains cycles

b) Acyclic Graph (DAG)

- No cycles

5. Special Types:

a) Tree

- Connected graph without cycles

- N nodes, N-1 edges

b) Complete Graph

- All vertices connected to all others

Task10:

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;

}

public class task10 {

    static class Edge {

        int src;

        int dest;

        Edge(int src, int dest) {

            this.src = src;

            this.dest = dest;

        }

    }

    static class Graph {

        int vertices;

        int edges;

        Edge[] edgeArray;

        Graph(int vertices, int edges) {

            this.vertices = vertices;

            this.edges = edges;

            edgeArray = new Edge[edges];

            System.out.println("Graph created with " + vertices +

                             " vertices and " + edges + " edges\n");

        }

        void addEdge(int index, int src, int dest) {

            edgeArray[index] = new Edge(src, dest);

        }

        void displayEdges() {

            System.out.println("Graph Edges:");

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

                System.out.println(edgeArray[i].src + " - " +

                                 edgeArray[i].dest);

            }

        }

    }

    public static void main(String[] args) {

        Graph graph = new Graph(5, 8);

        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.addEdge(7, 4, 5);

        graph.displayEdges();

    }

}

Output:

