****

**Submitted by**

Areef ur Rahman

L1F23BSSE0389

**Submitted To**

**Afifa Hameed**

Task\_1(University Students Record Management)

class Learner {

    int id;

    double marks;

    Learner leftChild, rightChild;

    public Learner(int id, double marks) {

        this.id = id;

        this.marks = marks;

        leftChild = null;

        rightChild = null;

    }

    @Override

    public String toString() {

        return " -------------------------\n" +

               " >> Roll No: " + id + "\n >> Score: " + marks;

    }

}

class MarksTree {

    public Learner rootNode;

    public Learner addLearner(Learner current, int id, double marks) {

        if (current == null) {

            System.out.println("Learner with ID (" + id + ") added!");

            return new Learner(id, marks);

        } else if (id < current.id) {

            current.leftChild = addLearner(current.leftChild, id, marks);

        } else if (id > current.id) {

            current.rightChild = addLearner(current.rightChild, id, marks);

        } else {

            System.out.println("Warning: Learner with ID " + id + " already exists!");

        }

        return current;

    }

    public boolean findLearner(Learner current, int id) {

        if (current == null) {

            System.out.println("Learner not found.");

            return false;

        }

        if (id == current.id) {

            System.out.println("Learner located!\nDetails:\n" + current);

            return true;

        } else if (id < current.id) {

            return findLearner(current.leftChild, id);

        } else {

            return findLearner(current.rightChild, id);

        }

    }

    public void printLearners(Learner current) {

        if (current != null) {

            printLearners(current.leftChild);

            System.out.println(current);

            printLearners(current.rightChild);

        }

    }

    public Learner getMinScoreLearner(Learner current) {

        if (current == null) return null;

        Learner min = current;

        Learner leftMin = getMinScoreLearner(current.leftChild);

        Learner rightMin = getMinScoreLearner(current.rightChild);

        if (leftMin != null && leftMin.marks < min.marks) {

            min = leftMin;

        }

        if (rightMin != null && rightMin.marks < min.marks) {

            min = rightMin;

        }

        return min;

    }

    public Learner getMaxScoreLearner(Learner current) {

        if (current == null) return null;

        Learner max = current;

        Learner leftMax = getMaxScoreLearner(current.leftChild);

        Learner rightMax = getMaxScoreLearner(current.rightChild);

        if (leftMax != null && leftMax.marks > max.marks) {

            max = leftMax;

        }

        if (rightMax != null && rightMax.marks > max.marks) {

            max = rightMax;

        }

        return max;

    }

}

public class Task1 {

    public static void main(String[] args) {

        MarksTree tree = new MarksTree();

        tree.rootNode = tree.addLearner(tree.rootNode, 1, 11);

        tree.rootNode = tree.addLearner(tree.rootNode, 2, 9);

        tree.rootNode = tree.addLearner(tree.rootNode, 5, 13);

        tree.rootNode = tree.addLearner(tree.rootNode, 4, 29);

        tree.rootNode = tree.addLearner(tree.rootNode, 3, 22);

        System.out.println("\nChecking duplicate entry:");

        tree.rootNode = tree.addLearner(tree.rootNode, 5, 14); // Duplicate ID

        System.out.println("\nLooking for learner with ID 3:");

        tree.findLearner(tree.rootNode, 3);

        System.out.println("\nList of all learners (sorted):");

        tree.printLearners(tree.rootNode);

        System.out.println("\n!! Learner with the lowest marks:");

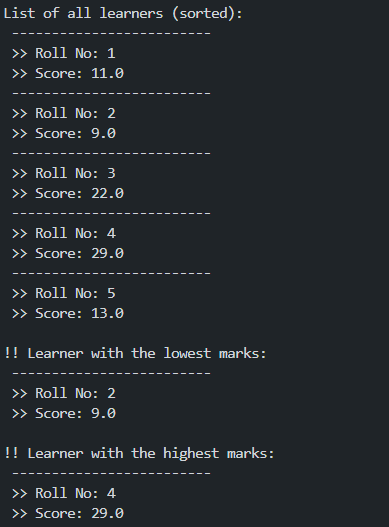
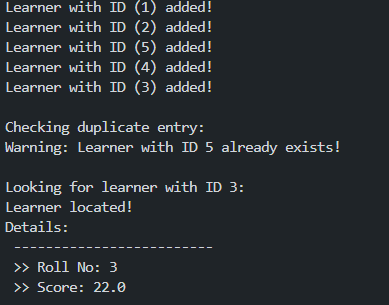
        System.out.println(tree.getMinScoreLearner(tree.rootNode));

        System.out.println("\n!! Learner with the highest marks:");

        System.out.println(tree.getMaxScoreLearner(tree.rootNode));

    }

}

**Output:**  


Task\_2 (Patients Record Management)

class Record {

    int id;

    String name;

    Record left, right;

    public Record(int id, String name) {

        this.id = id;

        this.name = name;

        left = right = null;

    }

    @Override

    public String toString() {

        return " ----------------------------\n" +

               " >> Patient ID: " + id + "\n >> Patient Name: " + name;

    }

}

class HospitalSystem {

    public Record head;

    public Record addPatient(Record head, int id, String name) {

        if (head == null) {

            System.out.println("Patient added: ID (" + id + "), Name: " + name);

            return new Record(id, name);

        } else if (id < head.id) {

            head.left = addPatient(head.left, id, name);

        } else if (id > head.id) {

            head.right = addPatient(head.right, id, name);

        } else {

            System.out.println("Duplicate ID! Patient with ID " + id + " already exists.");

        }

        return head;

    }

    public Record removePatient(Record head, int id) {

        if (head == null) return null;

        if (id < head.id) {

            head.left = removePatient(head.left, id);

        } else if (id > head.id) {

            head.right = removePatient(head.right, id);

        } else {

            if (head.left == null) return head.right;

            if (head.right == null) return head.left;

            head.id = findMin(head.right);

            head.right = removePatient(head.right, head.id);

        }

        return head;

    }

    int findMin(Record head) {

        int minId = head.id;

        while (head.left != null) {

            minId = head.left.id;

            head = head.left;

        }

        return minId;

    }

    public boolean findPatient(Record head, int id) {

        if (head == null) {

            System.out.println("No record found for the given ID.");

            return false;

        }

        if (id == head.id) {

            System.out.println("Record located:\n" + head);

            return true;

        } else if (id < head.id) {

            return findPatient(head.left, id);

        } else {

            return findPatient(head.right, id);

        }

    }

    public void showAllPatients(Record head) {

        if (head != null) {

            showAllPatients(head.left);

            System.out.println(head);

            showAllPatients(head.right);

        }

    }

}

public class Task2 {

    public static void main(String[] args) {

        HospitalSystem hs = new HospitalSystem();

        hs.head = hs.addPatient(hs.head, 1, "Ahsan");

        hs.head = hs.addPatient(hs.head, 5, "Farhan");

        hs.head = hs.addPatient(hs.head, 2, "Bilal");

        hs.head = hs.addPatient(hs.head, 4, "Tariq");

        hs.head = hs.addPatient(hs.head, 3, "Zubair");

        System.out.println("\nAll patient records (before deletion):");

        hs.showAllPatients(hs.head);

        System.out.println("\nRemoving patient with ID 4:");

        hs.head = hs.removePatient(hs.head, 4);

        System.out.println("\nAll patient records (after deletion):");

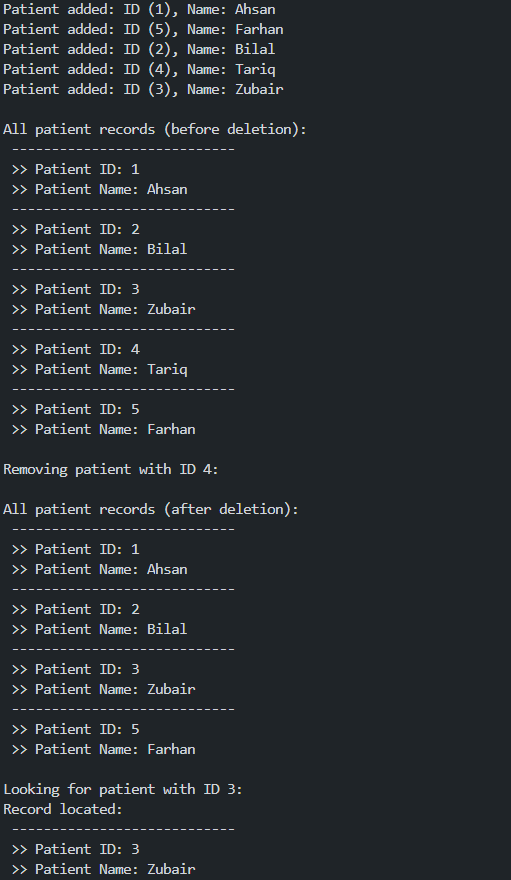
        hs.showAllPatients(hs.head);

        System.out.println("\nLooking for patient with ID 3:");

        hs.findPatient(hs.head, 3);

    }

}

**Output:**  


Task\_3 (“Book Store” Record Management)

class Node {

    int code;

    String name, writer;

    Node left, right;

    public Node(int code, String name, String writer) {

        this.code = code;

        this.name = name;

        this.writer = writer;

        this.left = null;

        this.right = null;

    }

    @Override

    public String toString() {

        return "--------------------------\n" +

               " Book Code: " + code +

               "\n Book Name: " + name +

               "\n Writer: " + writer;

    }

}

class BookTree {

    public Node start;

    public Node insert(Node start, int code, String name, String writer) {

        if (start == null) {

            System.out.println("Book added: " + code);

            return new Node(code, name, writer);

        } else if (code < start.code) {

            start.left = insert(start.left, code, name, writer);

        } else if (code > start.code) {

            start.right = insert(start.right, code, name, writer);

        } else {

            System.out.println("Book with code " + code + " already in the system!");

        }

        return start;

    }

    public boolean find(Node start, int code) {

        if (start == null) {

            System.out.println("No book found.");

            return false;

        }

        if (code == start.code) {

            System.out.println("Book located!\nDetails:\n" + start);

            return true;

        } else if (code < start.code) {

            return find(start.left, code);

        } else {

            return find(start.right, code);

        }

    }

    public void showBooks(Node start) {

        if (start != null) {

            showBooks(start.left);

            System.out.println(start);

            showBooks(start.right);

        }

    }

    public Node remove(Node start, int code) {

        if (start == null) {

            System.out.println("Cannot delete. Book with code " + code + " not found!");

            return null;

        }

        if (code < start.code) {

            start.left = remove(start.left, code);

        } else if (code > start.code) {

            start.right = remove(start.right, code);

        } else {

            if (start.left == null) {

                System.out.println("Deleted book with code: " + code);

                return start.right;

            } else if (start.right == null) {

                System.out.println("Deleted book with code: " + code);

                return start.left;

            }

            Node smallest = findSmallest(start.right);

            start.code = smallest.code;

            start.name = smallest.name;

            start.writer = smallest.writer;

            start.right = remove(start.right, smallest.code);

        }

        return start;

    }

    private Node findSmallest(Node start) {

        while (start.left != null) {

            start = start.left;

        }

        return start;

    }

}

public class Task3 {

    public static void main(String[] args) {

        BookTree bt = new BookTree();

        bt.start = bt.insert(bt.start, 10, "Learn Java Basics", "Ali Raza");

        bt.start = bt.insert(bt.start, 15, "Web Dev Essentials", "Sara Khan");

        bt.start = bt.insert(bt.start, 5, "Programming in C", "Ahmed Zubair");

        bt.start = bt.insert(bt.start, 12, "Database Intro", "Zain Aslam");

        bt.start = bt.insert(bt.start, 3, "Logic Building", "Maira Tariq");

        System.out.println("\nTrying to add a duplicate:");

        bt.start = bt.insert(bt.start, 5, "Oops Again", "Unknown");

        System.out.println("\nSearching for Book Code = 12:");

        bt.find(bt.start, 12);

        System.out.println("\nAll Available Books:");

        bt.showBooks(bt.start);

        System.out.println("\nRemoving Book Code = 15:");

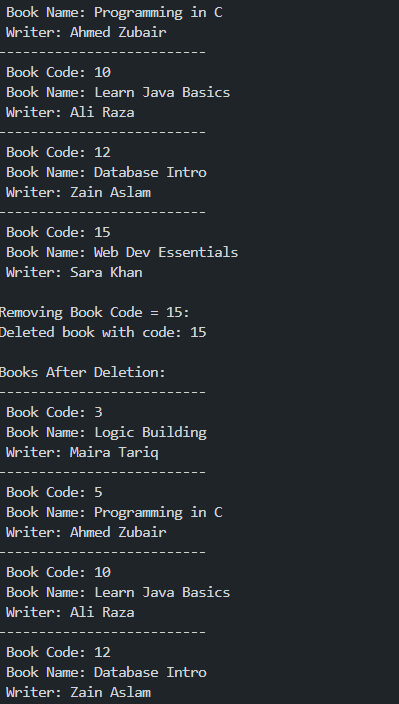
        bt.start = bt.remove(bt.start, 15);

        System.out.println("\nBooks After Deletion:");

        bt.showBooks(bt.start);

    }

}

**Output:**

Task\_4 (Packet Routing System Simulation)

class DeviceNode {

    int deviceCode;

    DeviceNode leftLink, rightLink;

    public DeviceNode(int code) {

        this.deviceCode = code;

        this.leftLink = null;

        this.rightLink = null;

    }

}

class DeviceManager {

    DeviceNode entryPoint;

    public DeviceNode addDevice(DeviceNode node, int code) {

        if (node == null) {

            return new DeviceNode(code);

        }

        if (code < node.deviceCode) {

            node.leftLink = addDevice(node.leftLink, code);

        } else {

            node.rightLink = addDevice(node.rightLink, code);

        }

        return node;

    }

    public void showInOrder(DeviceNode node) {

        if (node != null) {

            showInOrder(node.leftLink);

            System.out.print(">> Device ID: " + node.deviceCode + " | ");

            showInOrder(node.rightLink);

        }

    }

    public void showPreOrder(DeviceNode node) {

        if (node != null) {

            System.out.print("-- Starting Device ID: " + node.deviceCode + " | ");

            showPreOrder(node.leftLink);

            showPreOrder(node.rightLink);

        }

    }

    public void showPostOrder(DeviceNode node) {

        if (node != null) {

            showPostOrder(node.leftLink);

            showPostOrder(node.rightLink);

            System.out.print("~~ Turning Off Device ID: " + node.deviceCode + " | ");

        }

    }

}

public class Task4 {

    public static void main(String[] args) {

        DeviceManager manager = new DeviceManager();

        int[] codes = {45, 25, 65, 15, 35, 55, 75};

        for (int code : codes) {

            manager.entryPoint = manager.addDevice(manager.entryPoint, code);

        }

        System.out.println("=== Device Log View (In-Order Display) ===");

        manager.showInOrder(manager.entryPoint);

        System.out.println("\n");

        System.out.println("=== Device Boot Process (Pre-Order Display) ===");

        manager.showPreOrder(manager.entryPoint);

        System.out.println("\n");

        System.out.println("=== Power Off Sequence (Post-Order Display) ===");

        manager.showPostOrder(manager.entryPoint);

        System.out.println();

    }

}