```
import java.util.ArrayList;
import java.util.Scanner;
import java.util.Stack;

public class Node{

    Node left, right;
    int key;
    int height;

    Node(int key, int height, Node left, Node right){
        this.key = key;
        this.height = height;
        this.left = left;
        this.right = right;
    }
}

public class AVLTree {

    private Node root;


    public void insert(int key){
        root = insert(root, key);
    }

    private int findMin(){
        return findMin(root).key;
    }

    private Node findMin(Node x){
        if(x.left == null) return x;
        return findMin(x.left);
    }

    private int findMax(){
        return findMax(root).key;
    }

    private Node findMax(Node x){
        if(x.right == null) return x;
        return findMax(x.right);
    }

    private Node deleteMin(Node x){
        if(x.left == null) return x.right;
        x.left = deleteMin(x.left);
        return x;
    }

    private Node insert(Node x, int key){
        if(x == null) return new Node(key, 0, null, null);
        if(x.key < key) x.right = insert(x.right, key);
        else if(x.key > key) x.left = insert(x.left, key);
        x.height = Math.max(height(x.left), height(x.right)) + 1;
        return rebalance(x);
    }

    private Node lr(Node x){
```

```
    Node y = x.right;
    x.right = y.left;
    y.left = x;
    x.height = Math.max(height(x.left), height(x.right)) + 1;
    y.height = Math.max(height(y.left), height(y.right)) + 1;
    return y;
}

private Node rr(Node x){
    Node y = x.left;
    x.left = y.right;
    y.right = x;
    x.height = Math.max(height(x.left), height(x.right)) + 1;
    y.height = Math.max(height(y.left), height(y.right)) + 1;
    return y;
}

private Node rebalance(Node x){
    if(BB(x) > 1){
        if(BB(x.left) < 0){
            x.left = lr(x.left);
        }
        x = rr(x);
    } else if(BB(x) < -1){
        if(BB(x.right) > 0){
            x.right = rr(x.right);
        }
        x = lr(x);
    }
    return x;
}

private void delete(int key){
    root = delete(root, key);
}

private Node delete(Node x,int key){
  if(x == null) return null;
  if(x.key < key) x.right = delete(x.right, key);
  else if(x.key > key) x.left = delete(x.left, key);
  else {
      Node t = x;
      x = findMin(t.right);
      x.right = deleteMin(t.right);
      x.left = t.left;
  }
  x.height = Math.max(height(x.left), height(x.right)) + 1;
  return rebalance(x);
}

private int height(Node n){
    if(n == null) return -1;
    return n.height;
}

public int height(){
    return height(root);
}

public int BB(Node x){
```

```
    return height(x.left) - height(x.right);
}

public Iterable<Integer> keysInOrder(){
    ArrayList<Integer> items = new ArrayList<>();
    traversal(items, root);
    return items;
}

private void inorderTraversal(Node x){
    if(x == null) return;
    inorderTraversal(x.left);
    System.out.print(x.key + " ");
    inorderTraversal(x.right);
}

private void inorderTraversal(){
    inorderTraversal(root);
}

private void traversal(ArrayList<Integer> items, Node x){
    if(x == null) return;
    traversal(items, x.left);
    items.add(x.key);
    traversal(items, x.right);
}

public void printTree(){
    Stack<Node> globalStack = new Stack<>();
    globalStack.push(root);
    int nBlanks = 32;
    boolean isRowEmpty = false;
    System.out.println("......................................................");
    while(!isRowEmpty){
        Stack<Node> localStack = new Stack<>();
        isRowEmpty = true;
        for(int j = 0;j < nBlanks;++j){
            System.out.print(' ');
        }

        while(!globalStack.isEmpty()){
            Node temp = globalStack.pop();
            if(temp != null){
                System.out.print(temp.key);
                localStack.push(temp.left);
                localStack.push(temp.right);
                isRowEmpty = !(temp.left != null || temp.right != null);
            } else {
                System.out.print("--");
                localStack.push(null);
                localStack.push(null);
            }
            for(int j = 0;j < nBlanks * 2 - 2;++j)
                System.out.print(' ');
        }
        System.out.println();
        nBlanks /= 2;
        while(!localStack.isEmpty()) globalStack.push(localStack.pop());
    }
    System.out.println("......................................................");
```

```
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        AVLTree avlTree = new AVLTree();
        do{
            System.out.println("Enter integers and press any other keys with enter to stop: ");
            int x = input.nextInt();
            avlTree.insert(x);
            avlTree.printTree();
        } while(input.hasNextInt());
        for(int x : avlTree.keysInOrder())
            System.out.println(x);
    }
}
```