```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * binary tree diagram printer from the website
 * https://stackoverflow.com/questions/4965335/how-to-print-binary-tree-diagram
 */
class BTreePrinter {

    public static void printNode(Node root) {
        int maxLevel = BTreePrinter.maxLevel(root);

        printNodeInternal(Collections.singletonList(root), 1, maxLevel);
    }

    private static <T extends Comparable<?>> void printNodeInternal(List<Node>
nodes, int level, int maxLevel) {
        if (nodes.isEmpty() || BTreePrinter.isAllElementsNull(nodes))
            return;

        int floor = maxLevel - level;
        int endgeLines = (int) Math.pow(2, (Math.max(floor - 1, 0)));
        int firstSpaces = (int) Math.pow(2, (floor)) - 1;
        int betweenSpaces = (int) Math.pow(2, (floor + 1)) - 1;

        BTreePrinter.printWhitespaces(firstSpaces);

        List<Node> newNodes = new ArrayList<Node>();
        for (Node node : nodes) {
            if (node != null) {
                System.out.print(node.key);
                newNodes.add(node.left);
                newNodes.add(node.right);
            } else {
                newNodes.add(null);
                newNodes.add(null);
                System.out.print(" ");
            }

            BTreePrinter.printWhitespaces(betweenSpaces);
        }
        System.out.println("");

        for (int i = 1; i <= endgeLines; i++) {
            for (int j = 0; j < nodes.size(); j++) {
                BTreePrinter.printWhitespaces(firstSpaces - i);
                if (nodes.get(j) == null) {
                    BTreePrinter.printWhitespaces(endgeLines + endgeLines + i + 1);
                    continue;
                }

                if (nodes.get(j).left != null)
                    System.out.print("/");
                else
                    BTreePrinter.printWhitespaces(1);

                BTreePrinter.printWhitespaces(i + i - 1);
```

```
                if (nodes.get(j).right != null)
                    System.out.print("\\");
                else
                    BTreePrinter.printWhitespaces(1);

                BTreePrinter.printWhitespaces(endgeLines + endgeLines - i);
            }

            System.out.println("");
        }

        printNodeInternal(newNodes, level + 1, maxLevel);
    }

    private static void printWhitespaces(int count) {
        for (int i = 0; i < count; i++)
            System.out.print(" ");
    }

    private static <T extends Comparable<?>> int maxLevel(Node node) {
        if (node == null)
            return 0;

        return Math.max(BTreePrinter.maxLevel(node.left),
BTreePrinter.maxLevel(node.right)) + 1;
    }

    private static <T> boolean isAllElementsNull(List<T> list) {
        for (Object object : list) {
            if (object != null)
                return false;
        }

        return true;
    }
}

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;
    }
}

import java.util.ArrayList;
import java.util.Scanner;

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 printTree(){
    BTreePrinter.printNode(root);
}

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 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);
    }
}
```