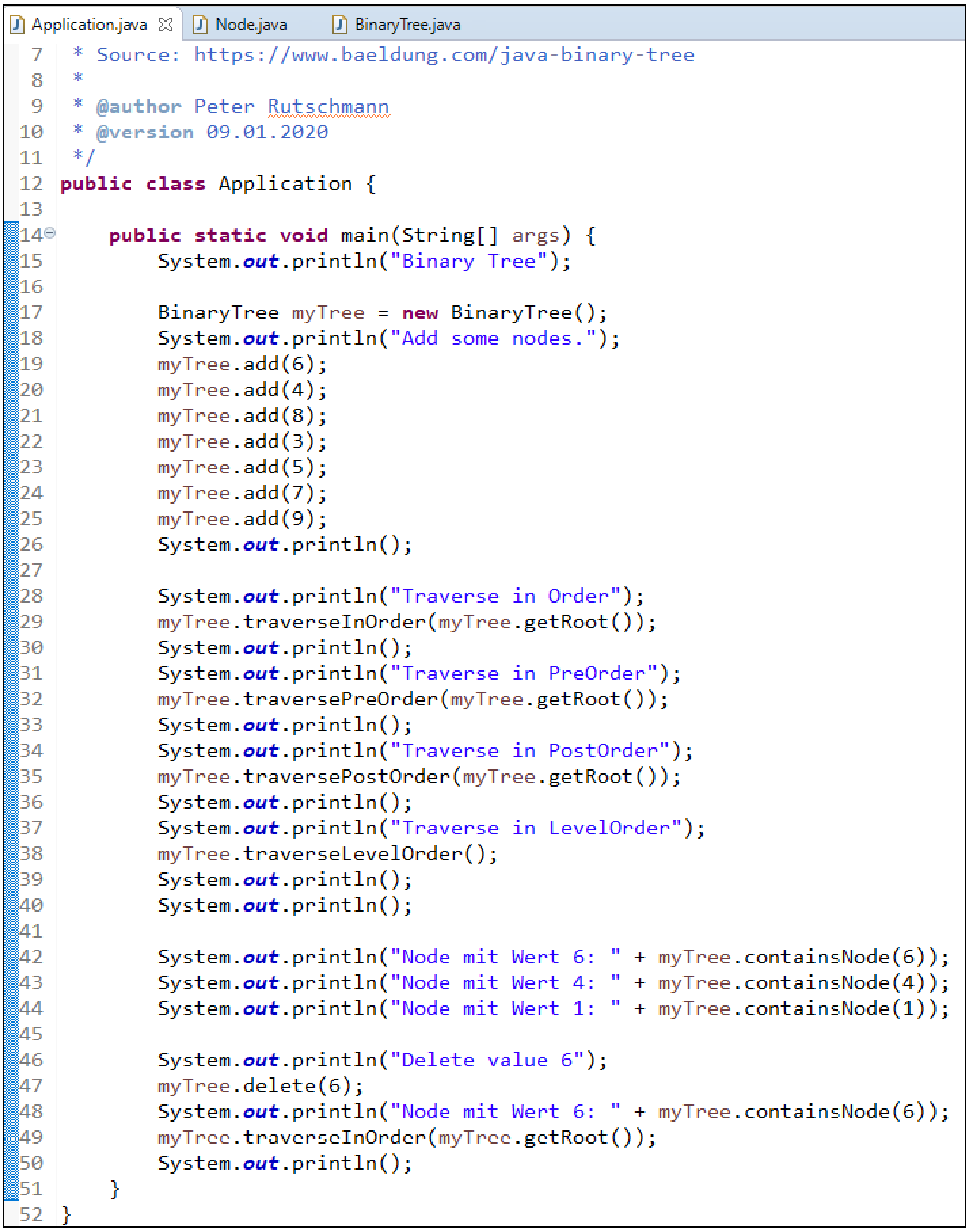
Binärer Suchbaum Implementierung Muster Lösung

Quelle:

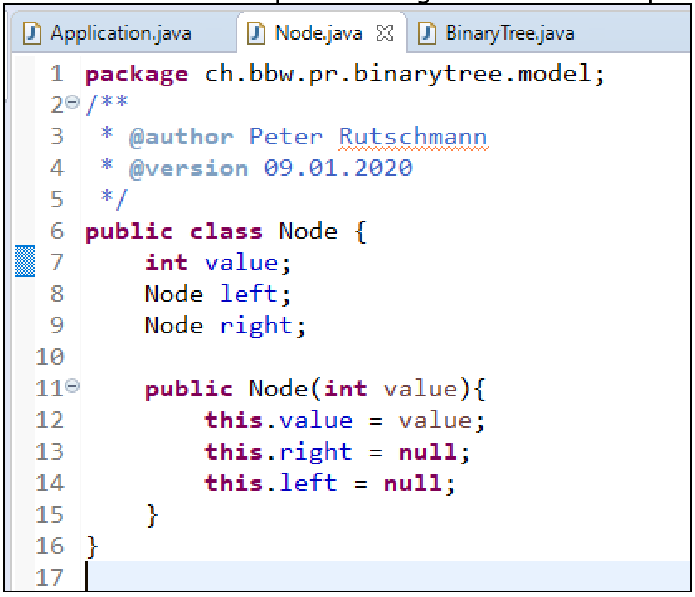
https://www.baeldung.com/java-binary-tree

Die Applikationsklasse habe ich selber implementiert.
Dafür auf die Tests aus dem Beispiel der Seite von Baeldung verzichtet.



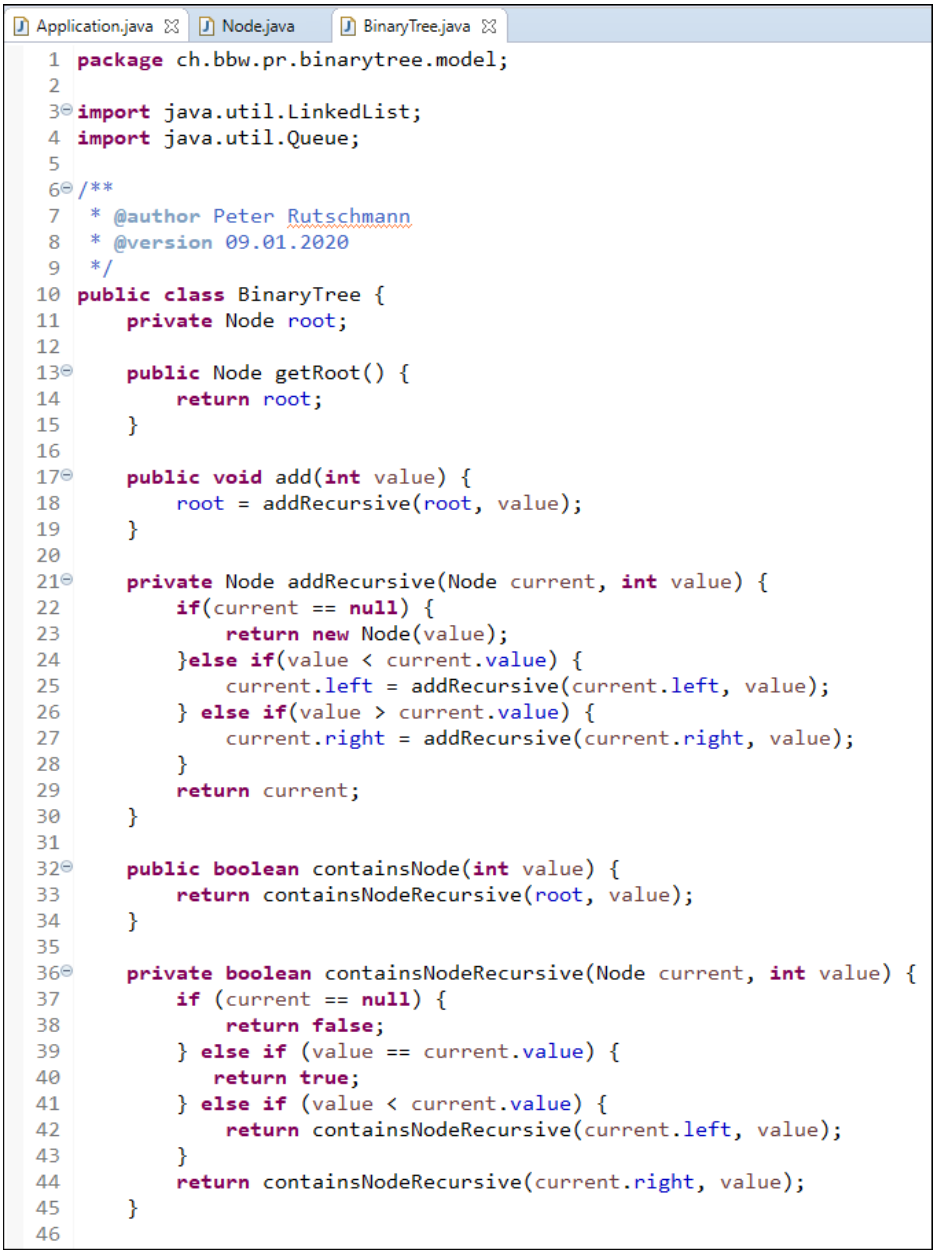
```java
 * Source: https://www.baeldung.com/java-binary-tree
 *
 * @author Peter Rutschmann
 * @version 09.01.2020
 */
public class Application {

    public static void main(String[] args) {
        System.out.println("Binary Tree");

        BinaryTree myTree = new BinaryTree();
        System.out.println("Add some nodes.");
        myTree.add(6);
        myTree.add(4);
        myTree.add(8);
        myTree.add(3);
        myTree.add(5);
        myTree.add(7);
        myTree.add(9);
        System.out.println();

        System.out.println("Traverse in Order");
        myTree.traverseInOrder(myTree.getRoot());
        System.out.println();
        System.out.println("Traverse in PreOrder");
        myTree.traversePreOrder(myTree.getRoot());
        System.out.println();
        System.out.println("Traverse in PostOrder");
        myTree.traversePostOrder(myTree.getRoot());
        System.out.println();
        System.out.println("Traverse in LevelOrder");
        myTree.traverseLevelOrder();
        System.out.println();
        System.out.println();

        System.out.println("Node mit Wert 6: " + myTree.containsNode(6));
        System.out.println("Node mit Wert 4: " + myTree.containsNode(4));
        System.out.println("Node mit Wert 1: " + myTree.containsNode(1));

        System.out.println("Delete value 6");
        myTree.delete(6);
        System.out.println("Node mit Wert 6: " + myTree.containsNode(6));
        myTree.traverseInOrder(myTree.getRoot());
        System.out.println();
    }
}
```

Die Nodeklasse entspricht weitgehend dem Beispiel von Baeldung

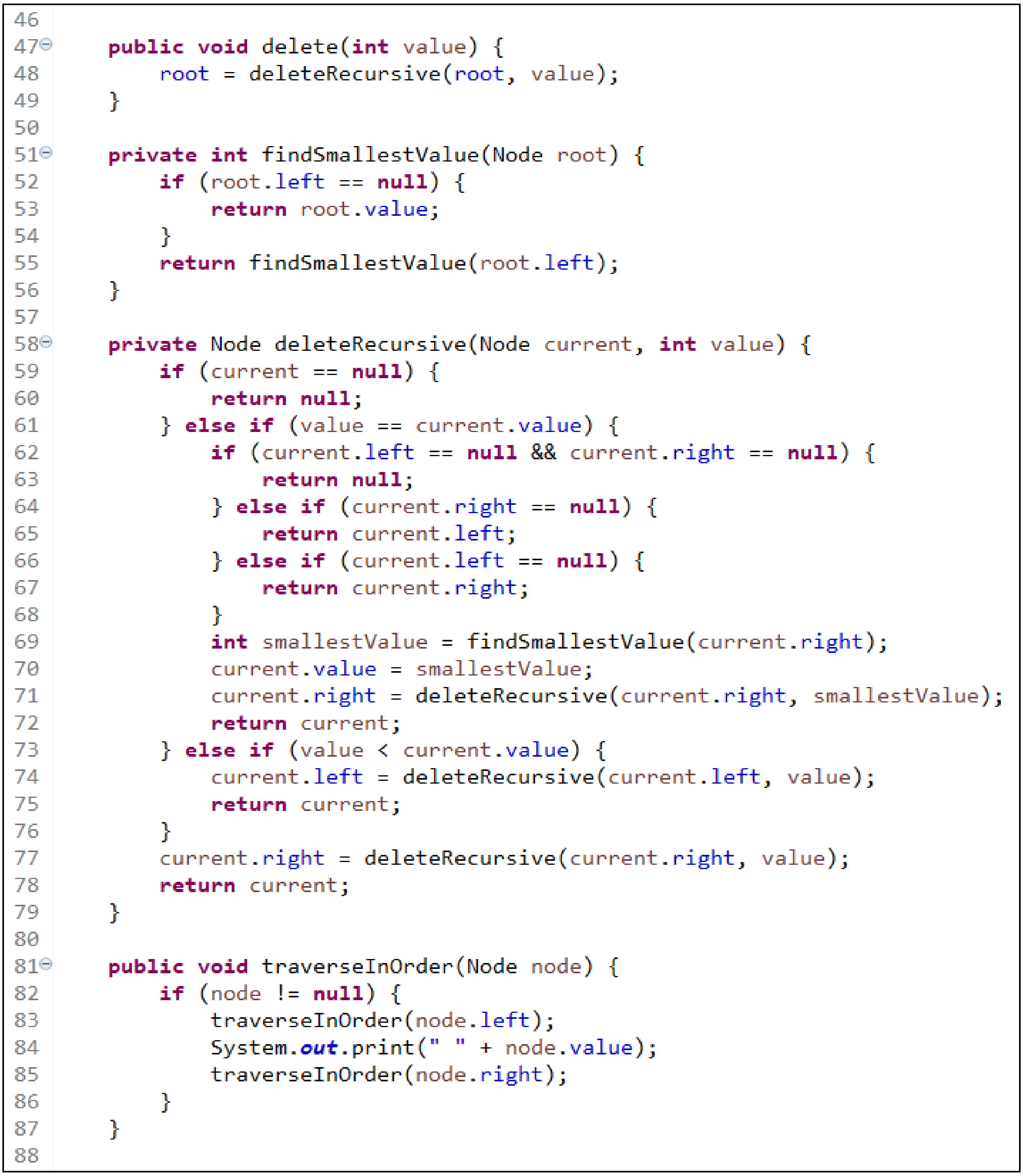


```java
package ch.bbw.pr.binarytree.model;
/**
 * @author Peter Rutschmann
 * @version 09.01.2020
 */
public class Node {
    int value;
    Node left;
    Node right;

    public Node(int value){
        this.value = value;
        this.right = null;
        this.left = null;
    }
}

```

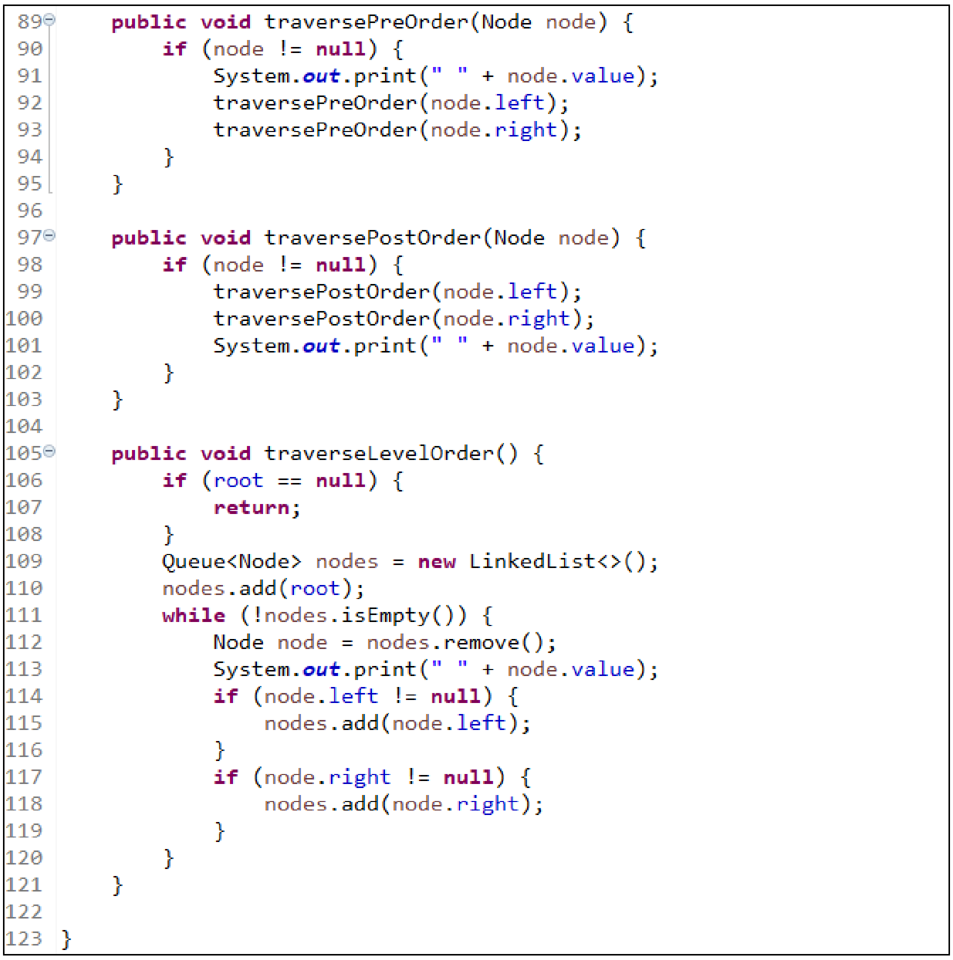
Die BinaryTree Klasse basiert auch auf den Methoden aus dem Beispiel von Baeldung.
Ich habe also nicht alles selber implementiert!!
Sie können sich viel Aufwand sparen!!



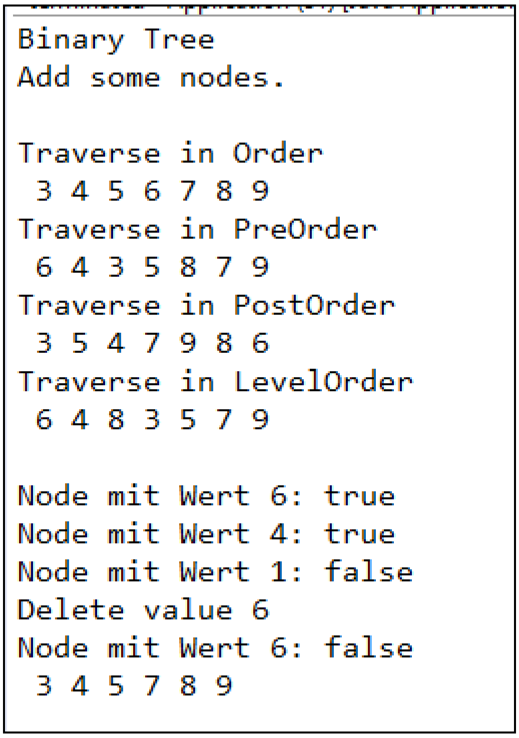
```java
package ch.bbw.pr.binarytree.model;

import java.util.LinkedList;
import java.util.Queue;

/**
 * @author Peter Rutschmann
 * @version 09.01.2020
 */
public class BinaryTree {
    private Node root;

    public Node getRoot() {
        return root;
    }

    public void add(int value) {
        root = addRecursive(root, value);
    }

    private Node addRecursive(Node current, int value) {
        if(current == null) {
            return new Node(value);
        }else if(value < current.value) {
            current.left = addRecursive(current.left, value);
        } else if(value > current.value) {
            current.right = addRecursive(current.right, value);
        }
        return current;
    }

    public boolean containsNode(int value) {
        return containsNodeRecursive(root, value);
    }

    private boolean containsNodeRecursive(Node current, int value) {
        if (current == null) {
            return false;
        } else if (value == current.value) {
           return true;
        } else if (value < current.value) {
            return containsNodeRecursive(current.left, value);
        }
        return containsNodeRecursive(current.right, value);
    }

```



```java
    public void delete(int value) {
        root = deleteRecursive(root, value);
    }

    private int findSmallestValue(Node root) {
        if (root.left == null) {
            return root.value;
        }
        return findSmallestValue(root.left);
    }

    private Node deleteRecursive(Node current, int value) {
        if (current == null) {
            return null;
        } else if (value == current.value) {
            if (current.left == null && current.right == null) {
                return null;
            } else if (current.right == null) {
                return current.left;
            } else if (current.left == null) {
                return current.right;
            }
            int smallestValue = findSmallestValue(current.right);
            current.value = smallestValue;
            current.right = deleteRecursive(current.right, smallestValue);
            return current;
        } else if (value < current.value) {
            current.left = deleteRecursive(current.left, value);
            return current;
        }
        current.right = deleteRecursive(current.right, value);
        return current;
    }

    public void traverseInOrder(Node node) {
        if (node != null) {
            traverseInOrder(node.left);
            System.out.print(" " + node.value);
            traverseInOrder(node.right);
        }
    }

```



```java
    public void traversePreOrder(Node node) {
        if (node != null) {
            System.out.print(" " + node.value);
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }

    public void traversePostOrder(Node node) {
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" " + node.value);
        }
    }

    public void traverseLevelOrder() {
        if (root == null) {
            return;
        }
        Queue<Node> nodes = new LinkedList<>();
        nodes.add(root);
        while (!nodes.isEmpty()) {
            Node node = nodes.remove();
            System.out.print(" " + node.value);
            if (node.left != null) {
                nodes.add(node.left);
            }
            if (node.right != null) {
                nodes.add(node.right);
            }
        }
    }

}
```



```
Binary Tree
Add some nodes.

Traverse in Order
 3 4 5 6 7 8 9
Traverse in PreOrder
 6 4 3 5 8 7 9
Traverse in PostOrder
 3 5 4 7 9 8 6
Traverse in LevelOrder
 6 4 8 3 5 7 9

Node mit Wert 6: true
Node mit Wert 4: true
Node mit Wert 1: false
Delete value 6
Node mit Wert 6: false
 3 4 5 7 8 9
```