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.

Application.java | Node.java | BinaryTree.java

```
 7  * Source: https://www.baeldung.com/java-binary-tree
 8  *
 9  * @author Peter Rutschmann
10  * @version 09.01.2020
11  */
12 public class Application {
13
14     public static void main(String[] args) {
15         System.out.println("Binary Tree");
16
17         BinaryTree myTree = new BinaryTree();
18         System.out.println("Add some nodes.");
19         myTree.add(6);
20         myTree.add(4);
21         myTree.add(8);
22         myTree.add(3);
23         myTree.add(5);
24         myTree.add(7);
25         myTree.add(9);
26         System.out.println();
27
28         System.out.println("Traverse in Order");
29         myTree.traverseInOrder(myTree.getRoot());
30         System.out.println();
31         System.out.println("Traverse in PreOrder");
32         myTree.traversePreOrder(myTree.getRoot());
33         System.out.println();
34         System.out.println("Traverse in PostOrder");
35         myTree.traversePostOrder(myTree.getRoot());
36         System.out.println();
37         System.out.println("Traverse in LevelOrder");
38         myTree.traverseLevelOrder();
39         System.out.println();
40         System.out.println();
41
42         System.out.println("Node mit Wert 6: " + myTree.containsNode(6));
43         System.out.println("Node mit Wert 4: " + myTree.containsNode(4));
44         System.out.println("Node mit Wert 1: " + myTree.containsNode(1));
45
46         System.out.println("Delete value 6");
47         myTree.delete(6);
48         System.out.println("Node mit Wert 6: " + myTree.containsNode(6));
49         myTree.traverseInOrder(myTree.getRoot());
50         System.out.println();
51     }
52 }
```

Die Nodeklasse entspricht weitgehend dem Beispiel von Baeldung

Application.java | Node.java | BinaryTree.java

```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!!

Application.java | Node.java | BinaryTree.java

```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);
    }

```

```
46
47      public void delete(int value) {
48          root = deleteRecursive(root, value);
49      }
50
51      private int findSmallestValue(Node root) {
52          if (root.left == null) {
53              return root.value;
54          }
55          return findSmallestValue(root.left);
56      }
57
58      private Node deleteRecursive(Node current, int value) {
59          if (current == null) {
60              return null;
61          } else if (value == current.value) {
62              if (current.left == null && current.right == null) {
63                  return null;
64              } else if (current.right == null) {
65                  return current.left;
66              } else if (current.left == null) {
67                  return current.right;
68              }
69              int smallestValue = findSmallestValue(current.right);
70              current.value = smallestValue;
71              current.right = deleteRecursive(current.right, smallestValue);
72              return current;
73          } else if (value < current.value) {
74              current.left = deleteRecursive(current.left, value);
75              return current;
76          }
77          current.right = deleteRecursive(current.right, value);
78          return current;
79      }
80
81      public void traverseInOrder(Node node) {
82          if (node != null) {
83              traverseInOrder(node.left);
84              System.out.print(" " + node.value);
85              traverseInOrder(node.right);
86          }
87      }
88
```

```
89      public void traversePreOrder(Node node) {
90          if (node != null) {
91              System.out.print(" " + node.value);
92              traversePreOrder(node.left);
93              traversePreOrder(node.right);
94          }
95      }
96
97      public void traversePostOrder(Node node) {
98          if (node != null) {
99              traversePostOrder(node.left);
100             traversePostOrder(node.right);
101             System.out.print(" " + node.value);
102         }
103     }
104
105     public void traverseLevelOrder() {
106         if (root == null) {
107             return;
108         }
109         Queue<Node> nodes = new LinkedList<>();
110         nodes.add(root);
111         while (!nodes.isEmpty()) {
112             Node node = nodes.remove();
113             System.out.print(" " + node.value);
114             if (node.left != null) {
115                 nodes.add(node.left);
116             }
117             if (node.right != null) {
118                 nodes.add(node.right);
119             }
120         }
121     }
122
123 }
```

```
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
```