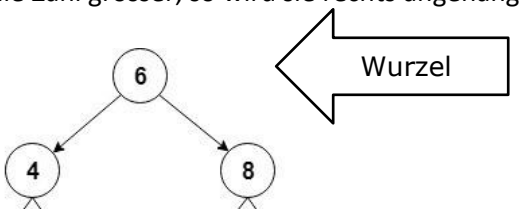
Binärer Suchbaum

(Quelle: Rinaldo Lanza, BBW)

Ein gebräuchlicher Typ von Binärbaum ist ein **binärer Suchbaum**.
Die Basis ist eine **Zahlenfolge, die in einen Baum sortiert eingefügt** wird.
Zum Beispiel die **unsortierte Zahlenfolge**:

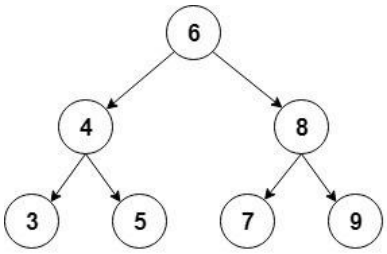
6 4 8 3 5 7 9

Beginnend mit 6 werden **alle Zahlen** der Reihe **nach in den ersten Knoten im Baum eingefügt**.
Der erste Knoten nennt man **Wurzel**.
Ist die Zahl kleiner, dann wird sie links angehängt.
Ist die Zahl grösser, so wird sie rechts angehängt.




```
bt.add(6); bt.add(4); bt.add(8); bt.add(3); bt.add(5); bt.add(7);
bt.add(9);
```

Es entsteht so dieser Baum:



Ein Vorteil ist nun, dass man den Baum relativ einfach nach einem Wert suchen kann.
Man weiss mit einem Vergleich immer, ob man nach links oder rechts gehen muss

Ziel

- Sie implementieren die Grundfunktionen für einen Binären Baum

Aufgabe Grundlage implementieren

Im Internet finden Sie ein gutes Beispiel für die Implementierung eines binären Suchbaumes, mit Erklärungen und mit Beispiel-Code:

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

Folgen Sie dem Beispiel von *Baeldung*.

Ablauf der Implementierung:

- Sie erstellen Sie die **Klasse Node**, sowie die **Grundklasse (BinaryTree)** mit einen **Wurzelknoten**.
- Dann folgt die Methode für das Einfügen eines Knoten:
  `private Node addRecursive(Node current, int value) { ... }`
- Die Seite von Baeldung aggiert mit Unittests, Sie können die Implementierung aber auch beispielhaft in der Main-Methode der Applikation realisieren.

```
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);
        //usw
```

- Eine weitere Methode prüft, ob ein Wert im Node vorhanden ist.
  `private boolean containsNodeRecursive(Node current, int value) {`

  Probieren Sie auch das der Applikation aus.

```
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));
```

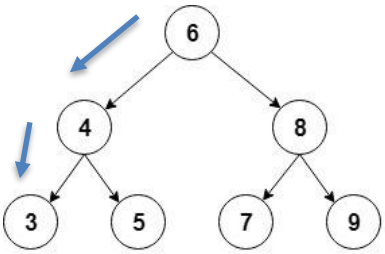
- Auf der Seite von Baeldung folgen weitere Methoden für das **Löschen eines Nodes** (nicht ganz einfach zu verstehen und für **das Traversieren** des ganzen Baumes.
  Dazu mehr auf den folgenden Seiten.

Baum traversieren

Im Kapitel 4 auf der Seite von *Baeldung* werden verschiedene Möglichkeiten zum Traversieren des Baums beschrieben.
Beim **Traversieren** geht es um das **Verfahren von Knoten zu Konten den ganzen Baum zu durchlaufen**.
Das kann zum Ausgeben des Ganzen Baumes (das machen die Beispiele von *Baeldung*) oder auch zum Suchen eines Knoten oder dessen Bearbeitung verwendet werden.
Je nach Art der Traversierung kann etwas rascher, effektiver gefunden werden.

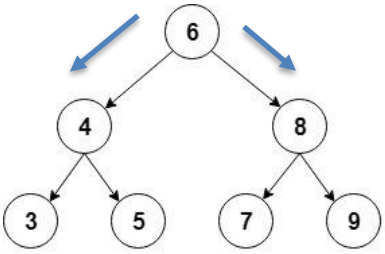
Tiefensuche:

Die Tiefensuche ist eine Art der Traversierung, die bei jedem Kind so tief wie möglich geht, bevor das nächste Geschwister untersucht wird.
Die Pfeile zeigen die ersten Schritte.



Breitensuche:

Die Breitensuche ist eine weitere Art der Traversierung, bei der zuerst alle Knoten einer Ebene besucht werden, bevor zur nächsten Ebene übergegangen wird.



Aufgabe:

Implementieren Sie zuerst die eine und danach die andere Traversierungsmethode. Erweitern Sie das Programm mit Ausgaben, um Licht ins Dunkle zu bringen.

```
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();
```

Knoten Löschen

Das **Löschen eines Knotens ist die komplexeste Aufgabe,** die es im Suchbaum zu implementieren gilt.
Dabei gibt es mehr Situationen, die mehr oder weniger komplex sind.

Löschen eines Blattes (Knoten ohne Kind-Knoten)
Die einfachste Situation, die man beim Löschen eines Knotens antreffen kann, ist ein **Knoten ohne Unterknoten**. Dieser Knoten kann einfach aus dem Baum gelöscht werden.

Löschen eines Knoten aus dem Baum (Knoten mit Kind-Knoten)
Löscht man einen Knoten aus dem Baum, so muss sein Platz von einem seiner *Kind-Knoten* eingenommen werden.
Das ist aufwendiger. Man muss sich entscheiden, welcher *Kind-Knoten* das ist.

Auf der Seite von *Baeldung* finden Sie dazu die Beispiele.

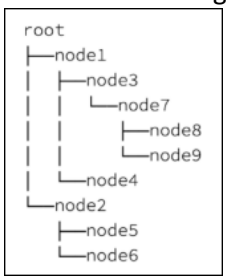
Aufgabe:
Programmieren Sie die Methode zum Löschen
Probieren Sie die Methoden aus.

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

Weiteres

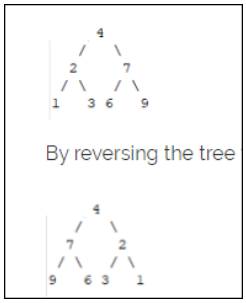
Auf der Seite von Baeldung finden Sie noch weiter spannende Themen:

Den Baum als Diagramm ausgeben.



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

Eine Tree "reversen"



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

Den Tree "Depth first search": https://www.baeldung.com/java-depth-first-search