

Bäume

- Ein Baum ist eine Datenstruktur, in dem sich Elemente hierarchisch anordnen lassen
- Die Elemente des Baumes nennt man **(Baum)knoten**
 - Ein Knoten kann mehrere **Kindknoten** haben, verbunden durch jeweils eine **Kante**
 - Jeder Knoten hat maximal einen **Elternknoten**
- Es gibt genau einen Knoten ohne Elternknoten, den **Wurzelknoten**/die **Wurzel**
- Knoten ohne Kinder heißen **Blattknoten** oder **Blätter**
- Die **Tiefe** eines Knotens ist sein Abstand zur Wurzel
 - Die Wurzel hat Tiefe 0
- Die **Höhe** eines Baums entspricht der maximal vorkommenden Tiefe

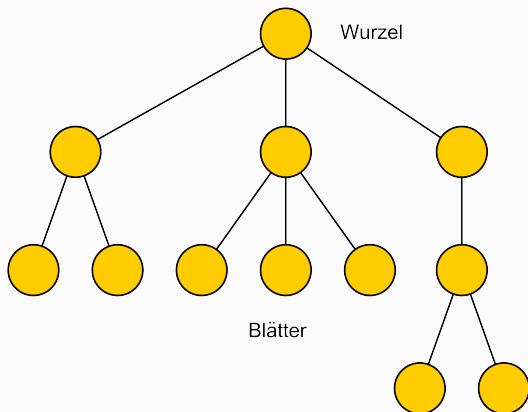


Abbildung 1: Beispiel eines Baumes

- Bäume tauchen an vielen Stellen in der Informatik auf:
 - Datenspeicherung für Suchen
 - Dokumentstrukturen (HTML/XML)
 - Verzeichnishierarchien
 - Menüstrukturen (Haupt- und Untermenüs)
 - ...

- Implementiere eine Klasse **BaumKnoten** zur Repräsentation eines einzelnen Baumknotens
 - Ein Knoten soll Daten speichern können
 - Basisfach: Ein Wert vom Typ **int**
 - Leistungsfach: Ein Wert von einem generischen Typ
 - Ein Knoten soll über Referenzen auf alle Kindknoten verfügen
 - Verwendung der Klasse **LinkedList** aus der Java-Klassenbibliothek
- Implementiere eine Klasse **BaumVerwalter** zur Erzeugung eines neuen Baums
 - Der Baum soll mindestens die Höhe drei haben
 - Es soll mindestens einen Knoten mit drei Kindern geben

- Um in allen Knoten eines Baumes zu suchen, bieten sich zwei unterschiedliche Verfahren an:
 - Tiefensuche
 - Breitensuche

1. Markiere die Wurzel als aktuell besuchten Knoten
2. Für den aktuell besuchten Knoten:
 - Falls es einen noch unbesuchten Kindknoten gibt, steige zu diesem ab (d. h. setze ihn als aktuell besuchten Knoten)
 - Falls nicht, gehe zum Elternknoten zurück

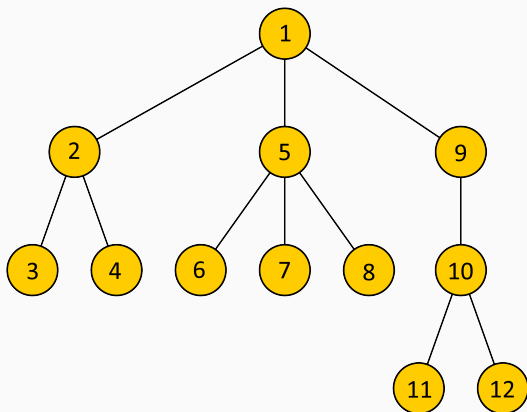


Abbildung 2: Besuchsreihenfolge bei der Tiefensuche

1. Beginne bei der Wurzel
2. Arbeite alle Kindknoten der Wurzel ab
3. Arbeite alle Kindknoten der in Schritt 2 besuchten Knoten ab
4. Arbeite alle Kindknoten der in Schritt 3 besuchten Knoten ab
5. usw...

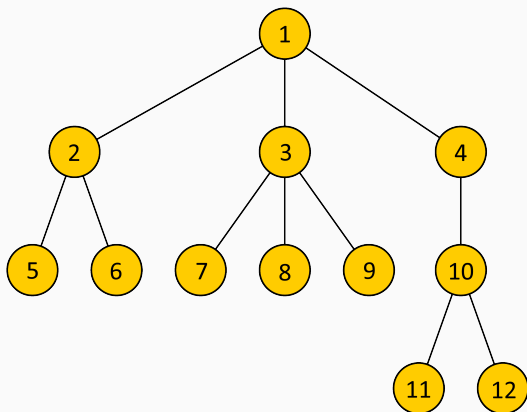


Abbildung 3: Besuchsreihenfolge bei der Breitensuche

- Implementiere Tiefensuche und Breitensuche als Methoden der Klasse **BaumVerwalter** aus Aufgabe 1
 - Die Methoden sollen jeweils beim Bearbeiten eines neuen Knotens dessen Wert ausgeben
- Tipps zur Implementierung
 - Tiefensuche: Rekursion
 - Breitensuche: Geordnete Liste der noch zu besuchenden Knoten führen

- Ein Baum kann als **rekursive Datenstruktur** aufgefasst werden
 - Jeder Knoten ist Wurzel eines **Teilbaums**
- Anstelle der Klasse **Baumknoten** wäre also auch eine Klasse **Baum** denkbar:

```
class Baum<E> {  
    private E value;  
    private LinkedList<Baum<E>> child;  
}
```