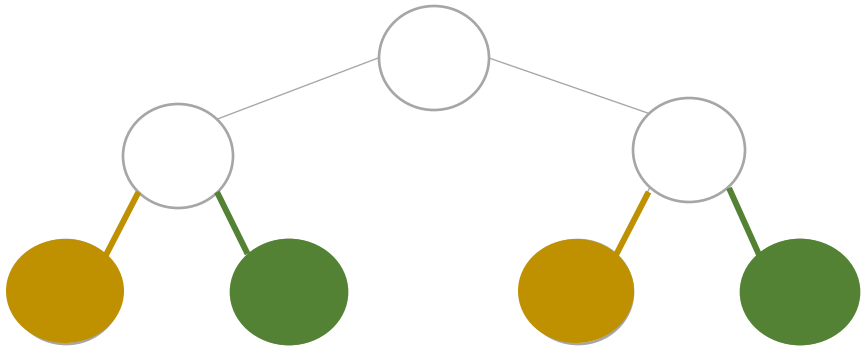
A04 Traverse Tree

public int **countWordsInSubTree** (Wort w)



1. Counter erstellen um diesen gegebenenfalls im weiteren Verlauf erhöhen
2. Wenn Wort w nicht null entspricht, wird der counter rekursiv um **w.getLeft** und **w.getRight** erhöht

Beispiel:
count + = countWordsInSubTree (w.getLeft());

3. Falls w = null ist, dann den Wert 0 retournieren

public int **getWordsWithPrefix** (String prefix)

1. Neues Set<String> erstellen
2. Hilfsmethode getAll aufrufen, um alle notwendigen Words in die ArrayList<Wort> **wordslist** zu laden
3. Die wordlist wird dann nach Wörtern durchsucht, die mit dem prefix starten

   Hierzu haben wird **word.getWort().startsWith(prefix)** verwenden
4. Falls dieses Kriterium zutrifft, wird dieses word.getWort() in das HashSet geladen

Beispiel:
„Ma“

wordslist:
„Hallo“, „Test“, „Maggie“, „Manjula“, „Bart“, „Burns“, „Marge“, „Anton“, „Bert“

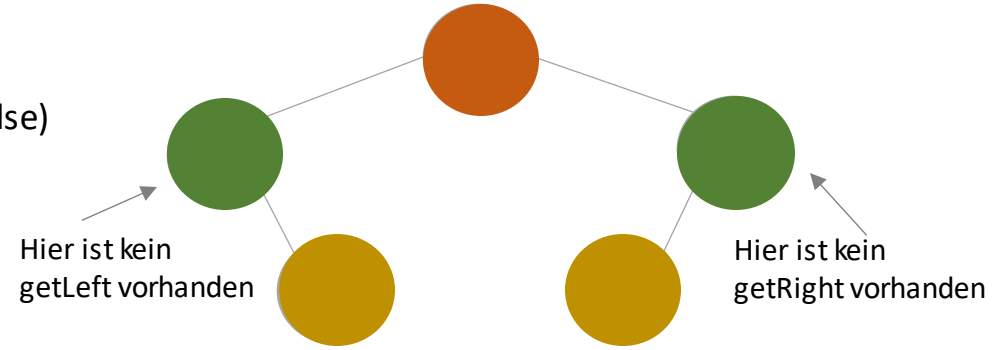
Set:
„**Ma**ggie“, „**Ma**njula“, „**Ma**rge“

Erzeugte Hilfsmethode:
public void **getAll** (Wort root)
Hier wird in die ArrayList<Wort> wordlist, ausgehend vom Root, rekursiv root.getLeft und root.getRight eingefügt.
Falls root == null, dann wird sofort ein return wiedergegeben

A05 Breitensuche

public List<Integer> getBreathFirstOrder (Node<Integer> start)

Zuerst wird eine ArrayList<Integer> result anlegt, um diese am Ende zu retournieren.
Der boolesche Ausdruck end, zeigt an, ob man bereits am Ende angekommen ist oder nicht (true/false)
In die ArrayList<Node<Integer>> nextEbene wird Node<Integer> start eingefügt



Solange das Ende noch nicht erreicht ist…

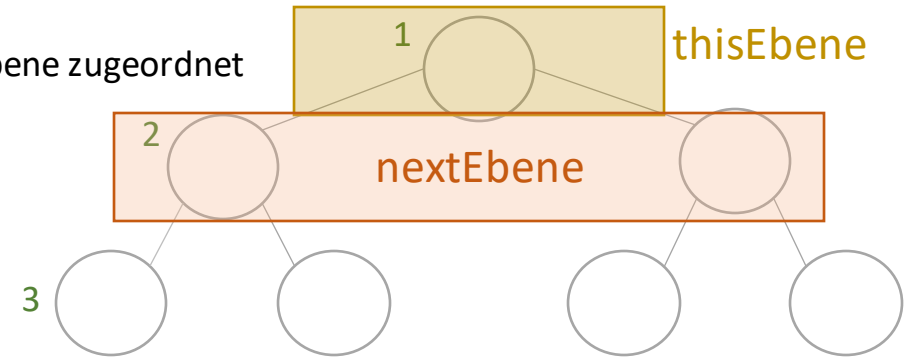
- wird eine neue ArrayList namens thisEbene angelegt, welcher die nextEbene zugeordnet wird,
- wird das Ende auf true gesetzt
- und durch thisEbene iteriert
  - Einerseits wird der Wert des Knotens eingefügt
  - Andererseits wird abgefragt, ob getLeft und getRight null entspricht (Falls nicht, wird dieser Knoten zur nextEbene hinzugefügt) und das Ende auf False gesetzt

---

public List<Integer> getBreathFirstOrderForLevel (Node<Integer> start, int level)

Zuerst wird eine List< Integer> namens result angelegt, um diese am Ende zu retournieren
Zur ArrayList<Node<Integer>> nextEbene wird start hinzugelügt und ein int current_level Counter auf 1 gesetzt), da Start Level 1 hat
Nun werden die Levels durchgegangen:
•Solange current_level kleiner/gleich level ist, entspricht die ArrayList thisEbene die ArrayList nextEbene zugeordnet
•Aus nextEbene wird eine neue ArrayList erzeugt
•Nun wird über die thisEbene iteriert, um die Kinder des Knotens in der nextEbene zu speichern
•Nachdem iteriert wurde wird der current_level Counter erhöht

Zuletzt werden die Werte von der thisEbene in die result-ArrayList eingefügt

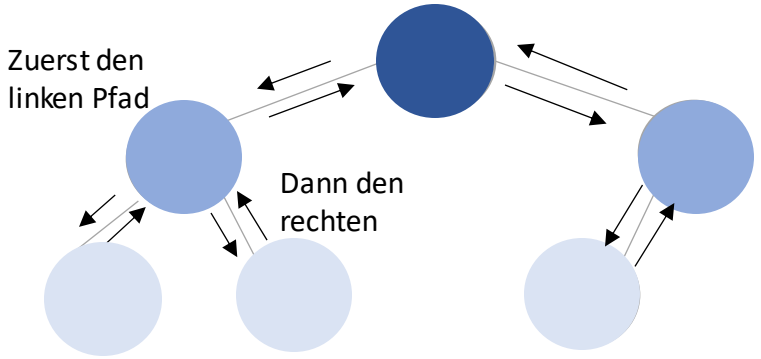


A06 Tiefensuche

public List<String> **getNodesInOrder** (Node<Film> node)

Zunächst wird eine List<String> result anlegt, um diese am Ende zu retournieren.
Vom Wurzelknoten aus, werden alle Knoten entlang eines Pfades untersucht. Zuerst werden die linken Knoten besucht, danach die rechten.
Wenn es keinen linken Knoten mehr gibt, geht man einen Knoten zurück und besucht dann die rechten Knoten. Falls die Kinderknoten != null sind, werden die Titel in unsere result List gespeichert.

Der Vorgang wird so lange wiederholt, bis man wieder an den Wurzelknoten gelangt.



public List<String> **getMinMaxPreOrder**(double min, double max)

Zuerst wird eine List< String> namens result angelegt, um diese am Ende zu retournieren.
Mithilfe der erzeugten getMinMaxPreOrder_calc Hilfsmethode, wird die Funktion rekursiv aufgerufen und fügt die Filmtitel (die das Kriterium erfüllen) in die result List ein.

<u>Erzeugte Hilfsmethode:</u>

public List<String> getMinMaxPreOrder_calc(Node<Film> node, double min, double max)

In der List<String> result wird der Filmtitel hinzugefügt, falls die Länge vom Film das min und max Kriterium nicht unter- bzw. überschreitet. Gleich wie oben, wird zuerst der linke dann der rechte Pfad besucht.