Prof. Dipl.-Inf. Ingrid Scholl | M.Eng. Torben Hensgens



# Praktikum 4 WS 15/16 IS 2

Thema: Binäre Suchbäume

***Allgemeine Hinweise:***

*Führen Sie im Allgemeinen immer dann Methoden ein, wenn dies sinnvoll ist.*

*Lesen Sie außerdem immer die zusätzlichen Hinweise (und weiterführende Links). Falls diese Hilfestellungen nicht ausreichen, um die geforderten Praktikumsaufgaben abzuschließen, sollten zusätzliche Informationen im Selbststudium bezogen werden.*

---

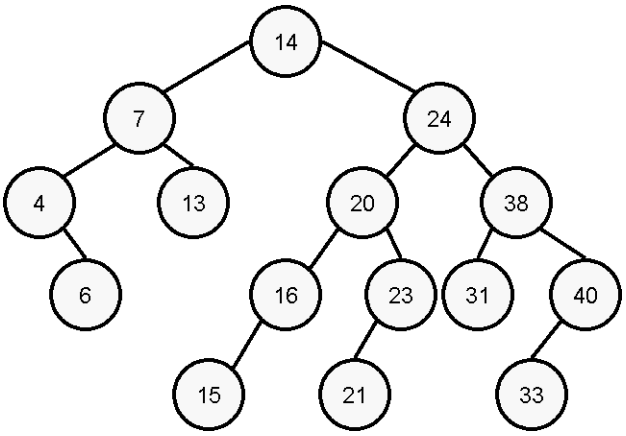
**Hinweis für beide Aufgaben**:

Bevor Sie auf ein Attribut (einer Instanz) einer Klasse zugreifen, testen Sie vorher, ob die Instanz ungleich null ist. Auf diese Weise vermeiden Sie viele NullPointerExceptions.

## Aufgabe 1: (Entwicklung eines binären Suchbaums)

Ein binärer Suchbaum ist eine Datenstruktur mit folgenden Eigenschaften:

- Die einzelnen Elemente der Datenstruktur werden als Knoten gespeichert. Ein Knoten hat dabei einen Wert und eine Reihe von Nachfolgerknoten. Dadurch ergibt sich eine Baumstruktur (die von oben nach unten wächst).
- Ein *binärer Baum* ist ein Baum, dessen Knoten maximal zwei Nachfolger haben.
- Die **Suchbaumeigenschaft** ist erfüllt, wenn die Werte der Knoten vergleichbar sind und für jeden Knoten des Baums gilt:
  - Die Werte aller Knoten des linken Teilbaums sind kleiner als der Wert des Knotens.
  - Die Werte aller Knoten des rechten Teilbaums sind größer als der Wert des Knotens.




Beispiel: Binärer Suchbaum mit 15 Knoten

Sie sollen in dieser Aufgabe einen binären Suchbaum entwickeln. Die Eingabewerte sollen dabei aus einer Datei eingelesen werden. Sie können davon ausgehen, dass die Eingabe keine doppelten Werte enthält, d.h. der resultierende Baum ist immer *duplikatfrei*.

# Praktikum 4 WS 15/16 IS 2

Thema: Binäre Suchbäume

Verwenden Sie die Klassen *Baum* und *Knoten*, die bereits einige Funktionen zur Verfügung stellen (diese Klassen befinden sich im ILIAS).

**a)** Erstellen Sie in der Klasse *Knoten* einen Konstruktor, der den *wert* eines neuen Knotens setzen kann und die Nachfolgerknoten (*links* und *rechts*) auf *null* setzt.

**b)** Schreiben Sie eine Methode (innerhalb der Klasse *Baum*), die einen neuen Knoten in den Baum einfügen kann. Die Methode soll folgende Signatur besitzen:

```
boolean einfuegen(int neu)
```

---

**Hinweis**:

Gehen Sie hierbei wie folgt vor:

- Überprüfen Sie zuerst, ob die Wurzel des Baumes *null* ist und fügen Sie in diesem Fall einen neuen Knoten als Wurzel ein.
- Andernfalls vergleichen Sie den Wert des aktuellen Knotens (zu Beginn ist der aktuelle Knoten == Wurzel) mit dem neuen Wert und machen eine Fallunterscheidung:
    1. Der neue Wert ist kleiner: Der linke Teilbaum des aktuellen Knoten muss überprüft werden. Ist dieser *null*, kann ein neuer Knoten mit dem neuen Wert hier eingefügt werden. Ist der linke Teilbaum nicht *null*, muss der Algorithmus weiterlaufen.
    2. Der neue Wert ist größer: Der rechte Teilbaum des aktuellen Knoten muss überprüft werden. Ist dieser *null*, kann ein neuer Knoten mit dem neuen Wert hier eingefügt werden. Ist der rechte Teilbaum nicht *null*, muss der Algorithmus weiterlaufen.

  Umschließen Sie den diesen Punkt mit einer Endlosschleife.

1. Falls ein neuer Knoten erfolgreich hinzugefügt wurde, soll die Methode *true* zurückgeben.

---

**c)** Schreiben Sie in der Klasse *Baum* eine Methode *printInorder()*, die den Baum in *Inorder* ausgibt. Wenn die Wurzel des Baums null ist, soll stattdessen „Leerer Baum." ausgegeben werden.

---

**Hinweis**:

*Inorder* bezeichnet die Regel, die angibt, in welcher Reihenfolge die Knoten des Baumes durchlaufen werden. Dabei werden für jeden Knoten folgende Schritte durchlaufen:

1. Zuerst wird der linke Teilbaum des Knotens in *Inorder* ausgegeben.
2. Dann folgt der Knoten selbst.
3. Schließlich wird der rechte Teilbaum des Knotens in *Inorder* ausgegeben.

Diese Funktion lässt sich **am einfachsten rekursiv** implementieren. Überprüfen Sie zunächst, ob die Wurzel des Baums null ist und arbeiten Sie dann die drei oben genannten Schritte ab.

---

**e)** Schreiben Sie eine main-Methode, in der Sie die *ReadFile*-Klasse aus Praktikum 1 verwenden, um die Werte aus der Datei „Knoten.txt“ auszulesen (die Datei liegt im ILIAS auf).

Erzeugen Sie ein Objekt der Klasse Baum. Iterieren Sie dann über die erhaltene *ArrayList* (z.B. mit einer *enhanced for loop*) und fügen Sie jedes Element in den Baum ein. Geben Sie anschließend den gesamten Baum mit `printInorder()` aus.

**Aufgabe 2: (Binäre Suche)**

Die *binäre Suche* wird verwendet, um ein (bestimmtes) Elemente effizient in einer Datenstruktur auffinden zu können.

Erweitern Sie die Klasse `Baum` aus Aufgabe 1 um folgende Methode:

```
boolean binSuche(int wert).
```

Die Funktion muss den zu suchenden Wert mit dem Wert des aktuellen Knotens vergleichen und im Fall, dass diese gleich sind, `true` zurückgeben, da der Wert in der Datenstruktur (hier: Binärer Suchbaum) gefunden wurde. Andernfalls durchsucht die Funktion entweder den linken oder den rechten Teilbaum des Knotens, je nachdem ob der zu suchende Wert kleiner oder größer ist.

---

**Hinweis**:

Es ist Ihnen freigestellt, ob Sie diese Funktion iterativ oder rekursiv implementieren.

---