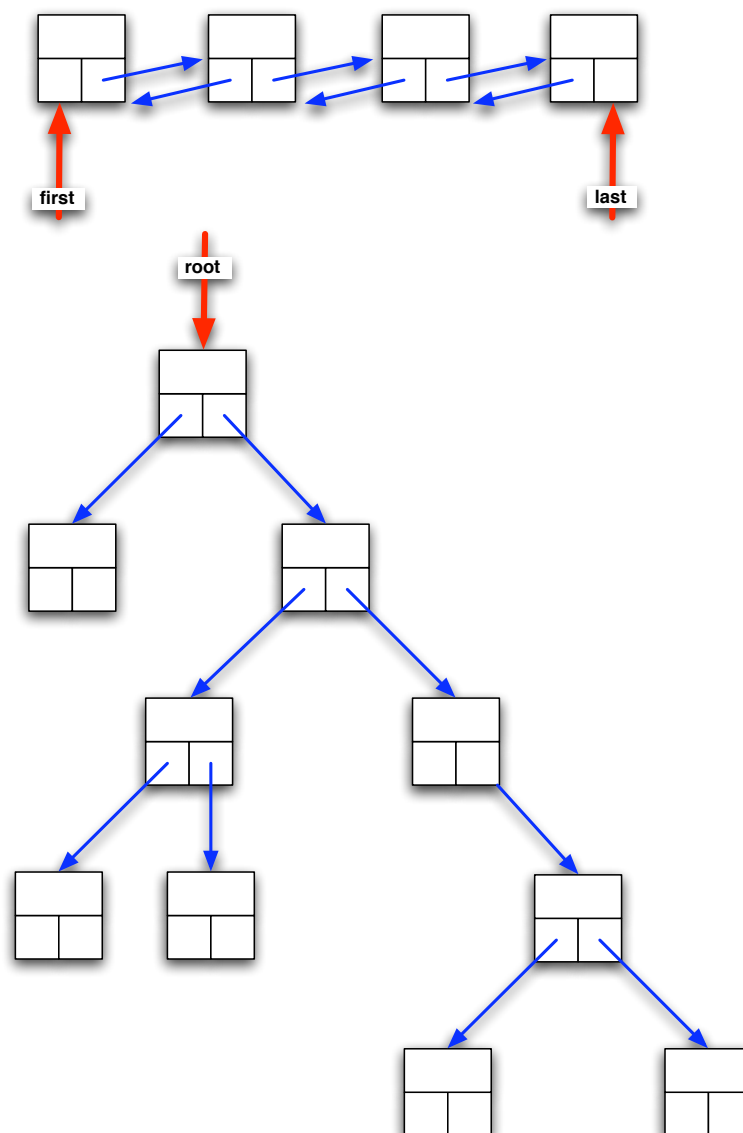


Folge 19 - Bäume

Grundlagen:

19.1 Binärbäume - Allgemeines

Unter **Bäumen** versteht man in der Informatik Datenstrukturen, bei denen jedes Element mindestens zwei Nachfolger hat. Bereits in der [Folge 17](#) haben wir ähnliche Datenstrukturen kennen gelernt, nämlich die doppelt verketteten Listen. Eine solche doppelt verkettete Liste besteht aus Elementen oder Knoten, die a) den eigentlichen Inhalt (zum Beispiel eine int-Zahl, einen String oder ein Objekt) und b) die Adresse des vorhergehenden und des nachfolgenden Elements (Knoten) speichern. Die Abbildung 19-1 zeigt im oberen Teil eine solche doppelt verkettete Liste.



19-1 Eine doppelt verkettete Liste und ein Binärbaum.

Unten in der Abbildung 19-1 ist ein so genannter **Binärbaum** gezeigt. Unter einem Binärbaum versteht man einen Baum, bei dem jeder Knoten (der Begriff „Knoten„ wird bei Bäumen häufiger benutzt als „Element„) exakt zwei Nachfolger hat, einen linken und einen rechten. Diese Nachfolger sind selbst wieder Binärbäume, wie die folgende rekursive Definition zeigt.

Binärbaum

Ein Binärbaum ist entweder leer oder besteht aus einer Wurzel mit einem linken und einem rechten Binärbaum als Nachfolger.

Auf den ersten Blick sieht diese Definition recht eigenartig aus, aber so ist das halt bei rekursiven Definitionen. Spielen wir die verschiedenen Fälle einmal durch.

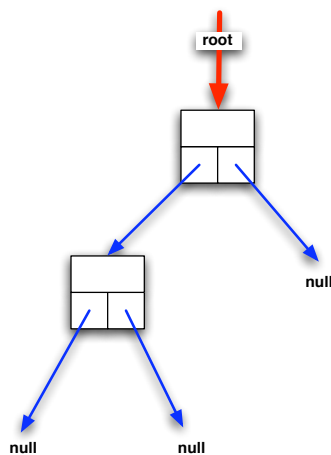
Fall 1: **Ein Binärbaum kann leer sein.** Zum besseren Verständnis schauen wir uns den Quelltext einer rudimentären Klasse **BinTree** an:

```
public class BinTree
{
    Element root;

    public BinTree()
    {
        root = null;
    }
    ...
}
```

Ein *leerer* Binärbaum liegt zum Beispiel dann vor, wenn ein Objekt der Klasse **BinTree** gerade erzeugt wurde. Das Attribut **root** des Objektes hat dann den Wert **null**.

Fall 2: **Ein Binärbaum besteht aus einer Wurzel mit einem rechten und einem linken Binärbaum.** Betrachten Sie dazu die Abbildung 19-2.



19-2 Ein kleiner Binärbaum.

Der obere Knoten des Binärbaums ist die **Wurzel**. Links an der Wurzel hängt ein weiterer Binärbaum, der seinerseits aus einer Wurzel und zwei Binärbäumen besteht, die aber beide leer sind. Rechts an der Wurzel hängt ein leerer Binärbaum.

Damit sollte Ihnen die rekursive Definition eigentlich klar geworden sein. Jetzt fehlen allerdings noch ein paar wichtige Fachbegriffe, auf die wir immer wieder zurückgreifen müssen.

Die Elemente eines Binärbaums werden auch als **Knoten** bezeichnet. Es gibt drei Typen von Knoten:

Wurzel

Ein Knoten, der keinen Vorgängerknoten hat. Die Wurzel ist der Ursprung des Baumes.

Innerer Knoten

Ein Knoten, der einen Vorgängerknoten und mindestens einen nicht leeren Nachfolgerknoten hat.

Blatt

Ein Knoten, der einen Vorgängerknoten und zwei leere Nachfolgerknoten hat.

Der Binärbaum in Abbildung 19-2 hat eine Wurzel und ein Blatt; innere Knoten sind in dem Baum aus Abbildung 19-1 zu sehen.

Grundlagen:

19.2 Binäre Suchbäume

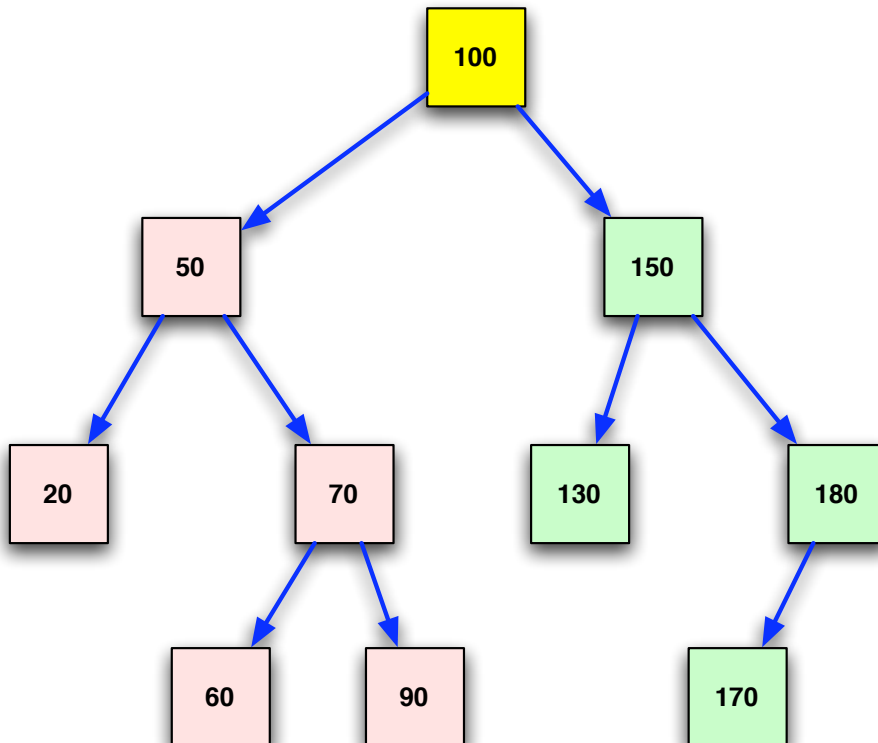
19.2.1 Grundlegendes, Rekursivität

Binärer Suchbaum

Ein **binärer Suchbaum** ist ein Binärbaum, bei dem für alle Unterbäume gilt:

1. der linke Unterbaum eines Knotens enthält nur Elemente, deren Wert kleiner als der Wert des Knotens ist.
2. der rechte Unterbaum eines Knotens enthält nur Elemente, deren Wert größer oder genauso groß wie der Wert des Knotens ist.

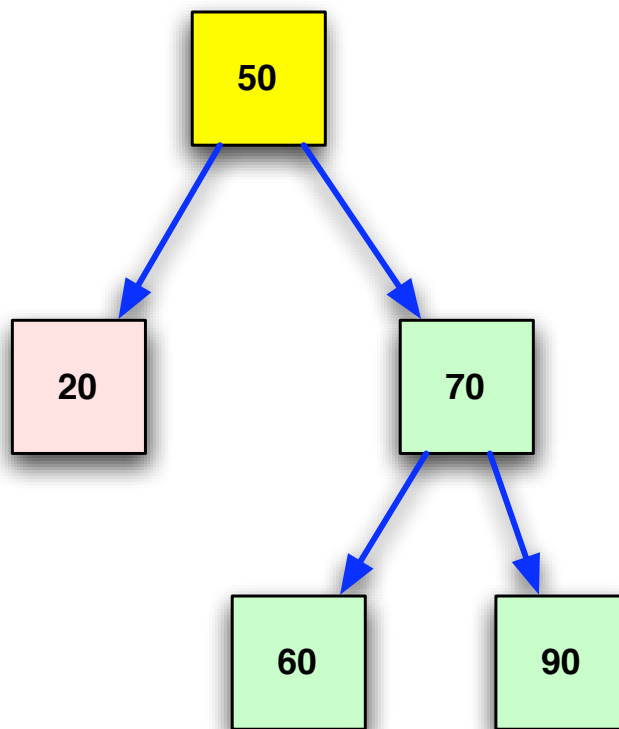
In der nächsten Abbildung sehen wir einen solchen binären Suchbaum.



19-3 Ein binärer Suchbaum.

Schauen wir uns mal den Wert der Wurzel an: 100. Alle Knoten des linken Unterbaums (rosa) haben einen Wert, der kleiner ist als 100. Alle Knoten des rechten Unterbaums (grün) haben einen Wert, der gleich oder größer als der Wert der Wurzel ist.

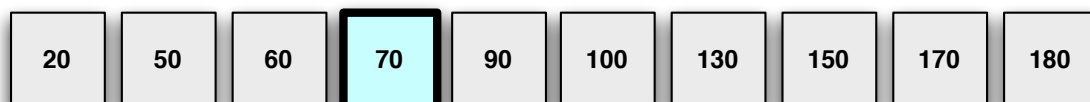
Der linke Unterbaum der Wurzel ist seinerseits ein binärer Suchbaum. Er besteht aus einer Wurzel (50), einem linken Teilbaum (der nur aus einer Wurzel besteht) und einem rechten Teilbaum (der wieder einen kompletten Binärbaum mit Wurzel und zwei Teilbäumen darstellt):



19-4 Der linke Unterbaum der 100 ist ebenfalls ein binärer Suchbaum, die Wurzel hat den Wert 50

19.2.2 Vorteile binärer Suchbäume

Welche Vorteile hat nun ein solcher binärer Suchbaum gegenüber einer doppelt verketteten Liste von Elementen? Schauen wir uns dazu eine konkrete Liste an. Auf die Wiedergabe der Zeiger wurde hier aus Übersichtsgründen verzichtet.



19-5 Eine Liste

Das Element 70 soll durch eine **lineare Suche** gefunden werden. Dazu sind vier Vergleiche notwendig, wie man unschwer erkennen kann. Die gleiche Suche in unserem binären Suchbaum aus Abbildung 19-3 würde nur drei Vergleiche kosten.

Um das Element 90 in der Liste zu finden, wären bei einer linearen Suche fünf Vergleiche notwendig. In dem Suchbaum finden wir die 90 nach vier Vergleichen - hier ist wieder ein kleiner Vorteil zu sehen.

Suchen wir jetzt nach der Zahl 100, so benötigen wir bei der Liste sechs Vergleiche, beim Suchbaum dagegen nur einen einzigen Vergleich - denn die 100 ist ja die Wurzel des Baums.

Übung 19.1 (2 Punkte)

- a) Ermitteln Sie sowohl für die Liste aus Abb. 19-5 wie auch für den Suchbaum aus Abb. 19-3 die durchschnittlichen Suchzeiten für die vorhandenen Elemente.
- b) Angenommen, nur 50% der gesuchten Element sind im Baum bzw. in der Liste vorhanden. Wie sehen jetzt die durchschnittlichen Suchzeiten aus?

Man erkennt sofort, dass das Suchen in dem Binärbaum im Durchschnitt viel schneller geht als in einer einfachen sortierten Liste. Würde man die sortierte Liste allerdings mit einem binären Verfahren durchsuchen, so ginge das genau so schnell wie das Durchsuchen eines Binärbaums - das binäre Suchen ist ja eine Methode, bei der man eine lineare sortierte Liste quasi vorübergehend in einen binären Suchbaum verwandelt

Aus wie vielen **Ebenen** (Schichten von Knoten) besteht ein Binärbaum mit 1000 Zahlen? Machen wir uns das mit Hilfe einer Tabelle klar:

Nummer der Ebene	Kapazität der Ebene	Kapazität des Baums
1	1	1
2	2	3
3	4	7
4	8	15
...		
N	$2^{(N-1)}$	$2^N - 1$

19-6 Kapazität eines binären Baumes

Ein Baum mit zehn Ebenen ($N = 10$) könnte also theoretisch $2^{10} - 1 = 1023$ Elemente speichern. Um 1000 Zahlen unterzubringen, benötigen wir also nur 10 Ebenen, wenn die Zahlen einigermaßen gleichmäßig verteilt sind. Wenn wir Pech haben, sind die Zahlen ungünstig verteilt, und wir brauchen 12, 14 oder sogar 20 Ebenen. Gehen wir einmal vom ungünstigen Fall aus, dass zum Speichern

der 1000 Zahlen 20 Ebenen benötigt werden. Das heißt, dass zum Finden einer beliebigen Zahl maximal 20 Vergleiche benötigt werden. Die durchschnittliche Zahl dürfte bei 18 oder weniger liegen.

Das Zeitverhalten des Suchens in einem binären Suchbaum ist sehr günstig, man kann es durch die Formel

$$O(N) = \log N$$

charakterisieren.

Zur Erinnerung: Bei der sortierten Liste aus 1000 Zahlen mussten wir durchschnittlich 500 mal vergleichen, um eine Zahl zu finden (lineares Suchen). Hier hatten wir das Zeitverhalten

$$O(N) = N$$

Der Binärbaum ist also hinsichtlich des Wiederfindens von Elementen mehr als 10 mal günstiger als die sortierte Liste. Bei noch größeren Zahlenmengen würde dieses Verhältnis sogar noch besser für den Binärbaum ausfallen.

Übung 19.2 (2 Punkte)

Zeichnen Sie auf ein Blatt Papier den binären Suchbaum, der sich ergibt, wenn man die folgenden Zahlen in genau dieser Reihenfolge eingibt:

300 - 200 - 100 - 50 - 120 - 70 - 130 - 400 - 500 - 600 - 187 - 34 - 100 - 50

Zeigen Sie Ihrem Lehrer den Baum. Er wird Ihnen dann eine weitere Zahl nennen, und Sie erklären ihm dann bitte, wo Sie diese Zahl einfügen wollen.

Praxis:

19.3 Binäre Suchbäume mit BlueJ

Jetzt wird es wieder einmal Zeit, sich an den Computer zu setzen und ein bisschen praktisch zu arbeiten. Als erstes programmieren Sie sich eine Klasse **Element** oder **Knoten**, und zwar auf die einfache Weise ohne Datenkapselung:

```
public class Element
{
    int value;
    Element left, right;

    public Element(int n)
    {
        value = n;
        left = null;
        right = null;
    }

    public void show()
    {
        System.out.println(„„+value);
    }
}
```

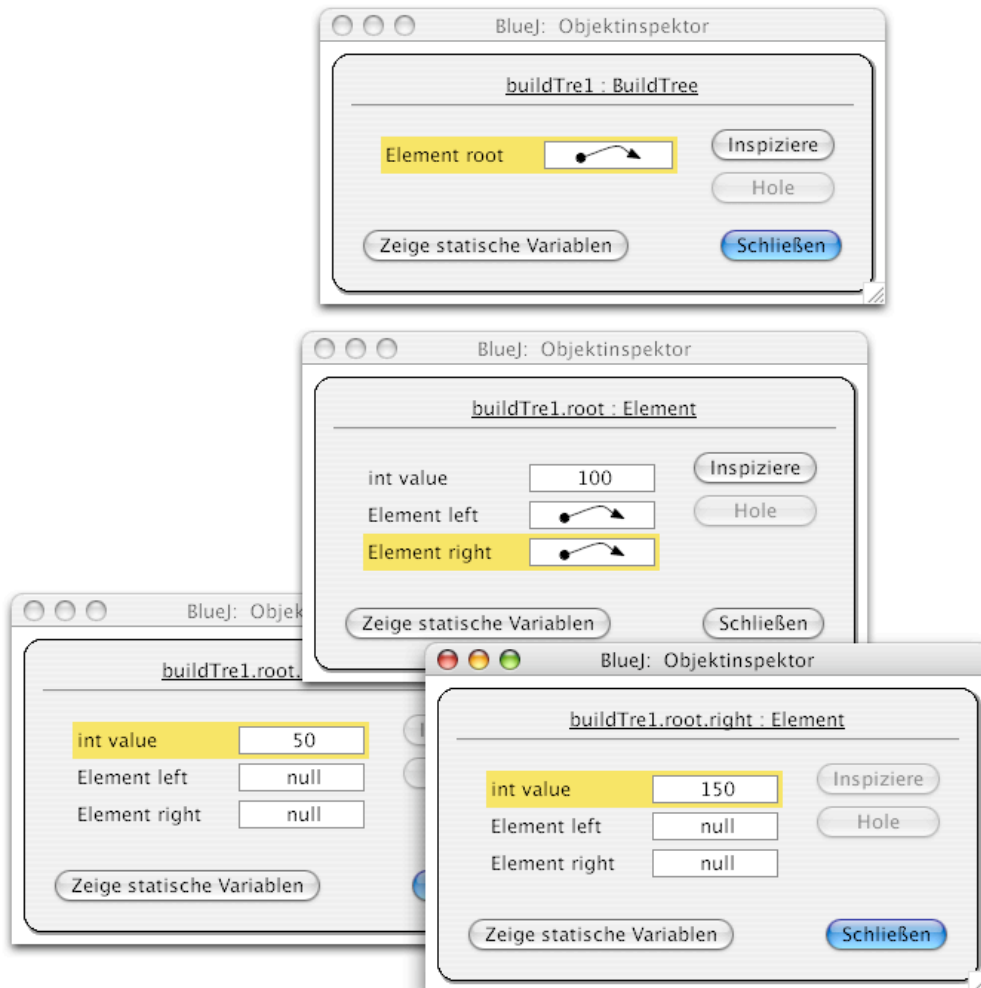
Diese Klasse speichert nur int-Zahlen. Um Objekte oder andere Datenstrukturen können wir uns später im Verlauf der Folge 19 noch kümmern, jetzt wollen wir uns auf das Wesentliche konzentrieren. Auf die Datenkapselung verzichten wir hier, um später den Zugriff auf die einzelnen Attribute wie **left** und **right** zu vereinfachen. Sonst müssten wir eigene sondierende Methoden schreiben - was an sich ja das korrekte Vorgehen wäre.

Als nächstes programmieren Sie bitte eine Klasse **BuildTree**, deren Aufgabe es ist, aus vielen solcher Elemente einen binären Suchbaum zu erstellen. Hier der vollständige Quelltext dieser Klasse:

```
public class BuildTree
{
    Element root;

    public BuildTree()
    {
        root = new Element(100);
        root.left = new Element(50);
        root.right = new Element(150);
    }
}
```

Nun schauen Sie sich mit dem **Objektinspektor** den erzeugten Baum einmal an. Der Baum wurde in der Tat genau so aufgebaut, wie es beabsichtigt wurde. Die Wurzel hat den Wert 100, der linke Nachfolger den Wert 50, und der rechte Nachfolger den Wert 150.



19-7 Ein einfacher Binärbaum wird mit dem Objektinspektor untersucht.

Übung 19.3 (2 Punkte)

Zeichnen Sie den Binärbaum, der sich ergibt, wenn folgende Methode ausgeführt wird:

```
public BuildTree()
{
    root = new Element(100);
    root.left = new Element(50);
    root.right = new Element(150);

    root.left.left = new Element(40);
    root.left.right = new Element(45);
    root.right.left = new Element(140);
    root.right.right = new Element(165);

    Element help = root.left.right;
    help.left = new Element(17);
    help.right = new Element(35);
    help = help.left;
    help.left = new Element(10);
}
```

Übung 19.4 (2 Punkte)

Verändern Sie den Konstruktor **BuildTree()** so, dass folgender Binärbaum aufgebaut wird, und zeigen Sie mit dem Objektinspektor, dass der Baum tatsächlich erzeugt wurde.

