

**Dozent**

Prof. Dr. Thomas Vetter  
 Departement  
 Mathematik und Informatik  
 Spiegelgasse 1  
 CH – 4051 Basel

**Assistenten**

Marcel Lüthi

**Tutoren**

Pascal Mafli  
 Loris Sauter  
 Linard Schwendener  
 Clemens Büchner  
 Florian Spiess  
 Jonathan Aellen  
 Lukas Stöckli

## Erweiterte Grundlagen der Programmierung (45398-01)

# Blatt 8

# [8 Punkte]

Vorbesprechung 12. Nov - 16. Nov  
 Abgabe 19. Nov - 23. Nov (vor dem Tutorat)

Wir empfehlen Ihnen, dass Sie im Buch “Sprechen Sie Java” bis Kapitel 13, sowie Kapitel 16 lesen, bevor Sie beginnen die Übungen zu lösen.

## Baum

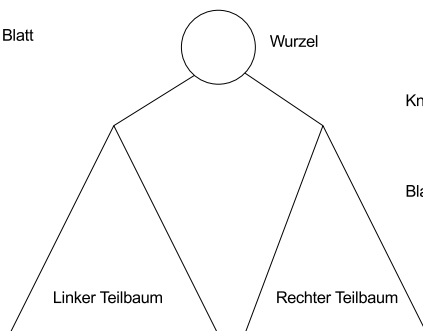
Ein Baum besteht aus Knoten und Blättern, und ist rekursiv definiert. Wir behandeln auf diesem Aufgabenblatt nur binäre Bäume, bei denen jeder Knoten zwei Kinder hat, aber das Konzept wird genauso auf n-äre Bäume erweitert. Der kleinste nichtleere Baum besteht aus einem Blatt. Grössere Bäume sind rekursiv definiert. Ein grösserer Baum besteht aus einem Knoten (der Wurzel), die zwei Bäume als Kinder enthält. Wir sprechen hier von dem linken und rechten Teilbaum. Die nachfolgende Abbildung veranschaulicht dieses Konzept.

Der kleinste nicht-leere Baum besteht aus einem einzelnen Blatt

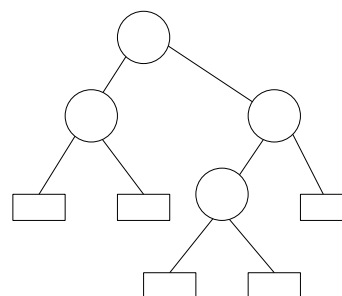
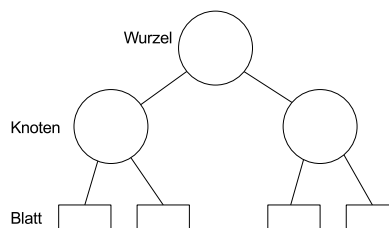


Blatt

Grössere Bäume sind rekursiv definiert. Sie enthalten einen Wurzelknoten und je einen linken und rechten Teilbaum.



Zwei Beispiele für binäre Bäume



## Binärer Suchbaum

Ein binärer Suchbaum ist eine dynamische Datenstruktur, die es ermöglicht Datensätze schnell aufzufinden. Er verwaltet eine Menge von Schlüssel-Wert Paare, wobei man anhand des Schlüssels nach dem Wert suchen kann. So kann man in einen Suchbaum z.B. eine Liste von Postleitzahlen mit dazugehörigen Orten speichern, und diese effizient auffinden.

Bei einem binären Suchbaum enthalten die inneren Knoten je einen Schlüssel und zwei Kind-Knoten. Dabei gilt, dass der Schlüssel des linken Kind-Knoten immer kleiner oder gleich dem in dem Knoten gespeicherten Schlüssel ist, der Schlüssel des rechten Kindknoten dagegen ist immer grösser. Jedes Blatt enthält zusätzlich noch den Wert, der zu dem Schlüssel gehört.

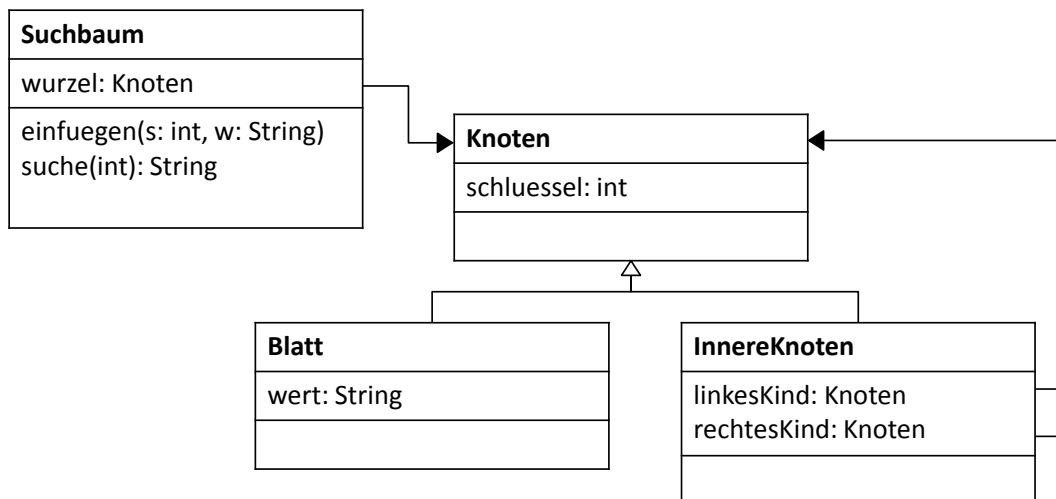
Wenn man in einem Suchbaum nach einem Schlüssel  $s_s$  sucht, muss man daher den gesuchten Schlüssel rekursiv mit den Schlüsseln  $s_i$  in den inneren Knoten vergleichen, und anhand des Ergebnisses entscheiden ob die Suche ihm linken Teilbaum oder im rechten Teilbaum fortgesetzt werden muss. Ist man an einem Blatt angelangt, vergleicht man den gesuchten Schlüssel  $s_s$  mit dem Schlüssel  $s_b$  im Blatt. Ist es der selbe Schlüssel, ist der gesuchte Wert im Blatt gespeichert, ansonsten ist der Wert nicht im Baum enthalten

Um in einen (anfangs leeren) Suchbaum einen Schlüssel einzufügen, verfährt man analog. Man sucht nach dem einzufügenden Schlüssel  $s_e$ , bis man an einem Blatt angelangt ist, das einen Schlüssel  $s_b$  hat. Nun ersetzt man das Blatt durch einen inneren Knoten. Der innere Knoten erhält den kleineren der Werte  $s_e, s_b$  als Schlüssel und zwei Kind-Knoten, welche die gefundenen und einzufügenden Schlüssel-Wert Paare enthalten.

### Aufgabe 1 - Suchbaum Implementation

[6 Punkte]

Implementieren Sie einen binären Suchbaum für Zahl-Zeichenketten Paare wie oben angegeben. Richten Sie sich bei der Implementierung nach diesem UML-Diagramm, die Konstruktoren sowie die öffentlichen *get*- und *set*-Methoden für die Felder müssen Sie selbst ergänzen.



Definieren Sie dazu:

- (a) eine Suchbaum-Klasse *Suchbaum*, die den eigentlichen Suchbaum enthält. Diese Klasse hat Funktionen zum Einfügen und zur Suche. Sie enthält zudem den Wurzelknoten des Suchbaumes. Geben Sie den Wurzelknoten über eine *getWurzel*-Funktion zurück. *Achtung: Der Name und die Signatur der Methoden einfuegen und suchen muss genau so wie im UML Diagramm angegeben sein, da ansonsten die Unit Tests nicht funktionieren.* [1 Punkt]
- (b) eine abstrakte Knoten-Klasse, die als Basisklasse für die inneren und die Blatt-Knoten dient. Diese Klasse enthält den Schlüssel sowie die dazu gehörige *getSchlüssel*-Methode. [1 Punkt]
- (c) eine Innere-Knoten-Klasse, die von der Knoten Klasse abgeleitet ist, und die zusätzlich zu dem Schlüssel noch zwei Kind-Knoten enthält. Definieren Sie für beide die getter/setter Methoden *getLinkesKind*, *getRechtesKind*, *setLinkesKind*, *setRechtesKind*. [1 Punkt]
- (d) eine Blatt-Klasse, die von der Knoten-Klasse abgeleitet ist, und die zusätzlich den zu dem Schlüssel gehörigen Wert enthält. Der Wert soll auch über eine Methode *getWert* zurück gegeben werden können. [1 Punkt]

Um Ihre Implementation zu testen schreiben Sie

- (e) eine Test-Klasse welche Schlüssel- Werte-Paare einfügt, nach Schlüssel sucht und den dazu gespeicherten Wert ausgibt. [1 Punkt]
  - (a) (3920, Zermatt)
  - (b) (3215, Gempenach)
  - (c) (4000, Basel)
  - (d) (4312, Magden)
  - (e) (7436, Medels)
  - (f) (3800, Interlaken)
- (f) für jede der Knoten-, Blatt- und Suchbaum-Klassen eine *toString* Methode so dass der Baum auf die Konsole ausgegeben werden kann. Verwenden Sie diese in der Test-Klasse um den Baum auszugeben. Versuchen Sie mit der Ausgabe die Struktur des Baumes zu veranschaulichen. [1 Punkt]

## Aufgabe 2 - Walker

[2 Punkte]

Wenn Sie von der Webseite die benötigten Klassen herunterladen finden Sie eine Klasse *Walker* im Paket *ch.unibas.informatik.cs101* welche sich ähnlich benutzen lässt wie Ihre *Turtle* Klasse. Der Konstruktor der Klasse *Walker* erwartet ein *ImageWindow* als Argument. Um die Klasse *Walker* zu benützen brauchen Sie folgende Anweisungen:

```
// imports for the two classes ImageWindow and Walker
import ch.unibas.informatik.cs101.ImageWindow;
import ch.unibas.informatik.cs101.Walker;
```

```
// creates a new ImageWindow
iw = new ImageWindow(widht, height);

// creates a new walker which will paint into iw
w = new Walker(iw);

// change between the two states: [draw / do not draw]
w.pressBallPen();

// turn the direction by an angle of 45.59 degrees (clockwise)
w.turn(45.59);

// move the walker 3.1415 pixels in the actual direction
w.move(3.1415);

// set the color for all future drawings to (RGB)=(180/180/255)
w.setColor(180, 180, 255)

// set the walker to the specified position and direction
w.setPos(5.0, 5.0);
w.setDir(1.0, 0.0);
```

- (a) Schreiben Sie eine Klasse KochWalk die eine *rekursive* Funktion besitzt, welche folgende Pfade erzeugen kann:



Rekursionstiefe 1



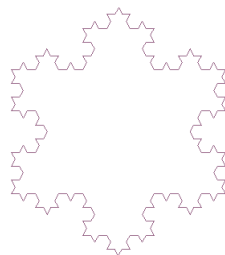
Rekursionstiefe 2



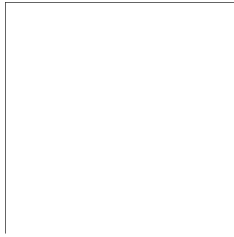
Rekursionstiefe 3

Verwenden Sie die Methoden `setPos` und `setDir` nicht in der rekursiven Funktion. Verwenden Sie diese Funktionen nur um zu Beginn die Startposition so zu setzen, dass die ganze Kurve im sichtbaren Bereich des ImageWindow gezeichnet wird. Schreiben Sie das Program so, dass die Rekursionstiefe als Eingabeparameter übergeben werden kann. Beachten Sie, dass Sie bei der Drehung positive und negative Zahlen verwenden können, die zurückzulegende Distanz jedoch immer positiv sein muss. [ $\frac{1}{2}$  Punkt]

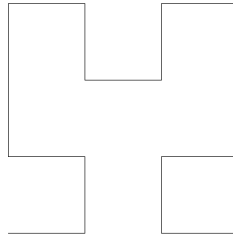
- (b) Schreiben Sie eine Methode welche die Rekursion 3 mal aufruft um eine Schneeflocke zu zeichnen. Schreiben Sie das Programm so, dass die Rekursionstiefe als Eingabeparameter übergeben werden kann. [ $\frac{1}{2}$  Punkt]



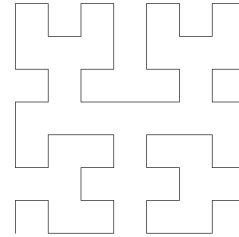
- (c) Schreiben Sie eine Klasse `HilbertWalk` welche eine rekursive Methode aufruft, und folgende Bilder erzeugt:



Rekursionstiefe 1



Rekursionstiefe 2



Rekursionstiefe 3

Es reicht wenn Sie diese Kurve für eine fixe Kantenlänge von 5 Pixel zeichnen. Eine sich der Rekursionstiefe anpassende Kantenlänge ist nicht nötig für die volle Punktzahl. Schreiben Sie das Program so, dass die Rekursionstiefe als Eingabeparameter übergeben werden kann. [1 Punkt]