Vergleich: List und BinaryTree

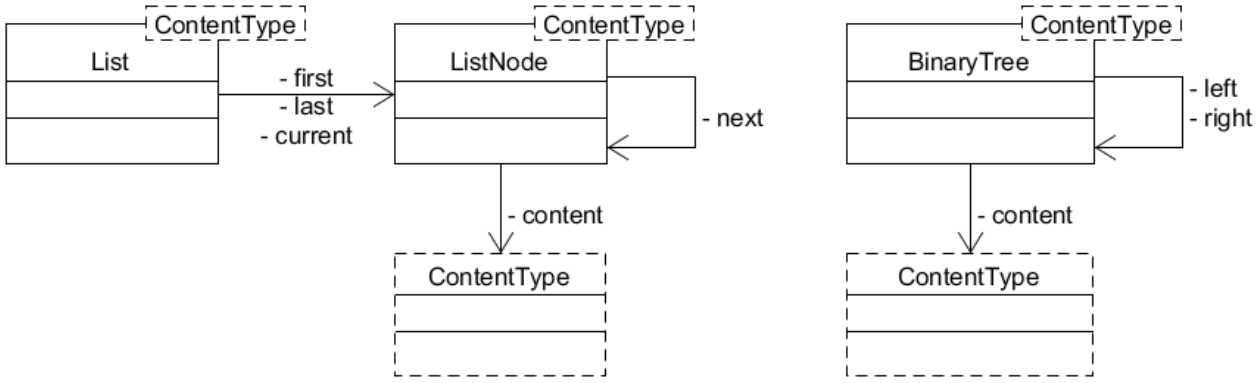
Mit der Klasse **BinaryTree** können Binärbäume erzeugt werden. In dieser Datenstruktur hat jeder Knoten (bis zu) zwei Nachfolger, einen linken und einen rechten. Jeder Nachfolger kann wieder zwei Nachfolger haben, und so weiter.

Wie bei den Klassen Queue, Stack und List gibt es auch hier eine rekursive Assoziation:
Objekte der Klasse BinaryTree „haben“ jeweils zwei Objekte der Klasse BinaryTree.

Die Datenstrukturen Queue, Stack und List bestehen in der Implementierung des Schulministeriums NRW aus je **zwei Klassen**, z.B. List und ListNode. ListNode-Objekte stellen die Knoten dar, aber die Methoden zum Einsatz der Liste in anderen Klassen werden von der Klasse List bereitgestellt. ListNode-Objekte oder Methoden der Klasse ListNode sind außerhalb der Klasse List nicht sichtbar, und werden für die Anwendung der Liste auch nicht benötigt.

Der Binärbaum wird hingegen nur durch **eine Klasse** implementiert:
Objekte der Klasse BinaryTree stellen die Knoten dar. Die Methoden der Klasse BinaryTree sind auch die Schnittstelle für andere Programmierer. Der Grund dafür ist, dass viele Algorithmen für Binärbäume rekursiv arbeiten. In einer Rekursion kann man jeden beliebigen Knoten eines Binärbaums (mit seinen Nachfolgern) auch wieder als Binärbaum verstehen, und darauf die Methoden der Klasse BinaryTree anwenden.

Die folgenden Klassendiagramme zeigen die Klassen für Liste und Binärbaum im Vergleich:



Objekte der Klasse BinaryTree

Wie List ist auch die Klasse BinaryTree „generisch“. Man kann also Objekte beliebiger Klassen als Inhalt mit den Knoten verbinden. Welche Klasse für die Inhalte verwendet wird, legt man bei der Deklaration eines BinaryTree-Objekts fest. Wie bei der Liste kann man innerhalb eines Binärbaums nur Objekte einer Klasse verwenden.

Jedes BinaryTree-Objekt hat genau ein Inhalts-Objekt, sowie zwei Referenzen auf weitere BinaryTree-Objekte – left und right, für den linken bzw. rechten Nachfolge-Knoten.

Ein „**leerer Knoten**“ hat kein Inhalts-Objekt und auch keine Nachfolger.
Sobald ein BinaryTree-Objekt jedoch ein Inhalts-Objekt hat, werden automatisch zwei weitere BinaryTree-Objekte als dessen Nachfolger erzeugt (die dann zunächst leere Knoten sind).

Methoden der Klasse BinaryTree

`BinaryTree()`
Erzeugt einen **leeren Knoten**. Die Referenzen left, right und content werden auf null gesetzt.

`BinaryTree(ContentType pContent)`
Erzeugt einen Knoten mit dem Inhalt „pContent“. Dieser Knoten ist also nicht leer.
Für die Referenzen left und right wird jeweils ein neuer leerer Knoten erzeugt.
Ausnahme: Falls pContent null ist, wird ein leerer Knoten erzeugt.

`BinaryTree(ContentType pContent, BinaryTree pLeft, BinaryTree pRight)`
Erzeugt einen Knoten mit dem Inhalt „pContent“.
Zusätzlich werden zwei bereits bestehende Binärbäume (pLeft und pRight) als linker bzw. rechter Teilbaum an den neuen Knoten gehängt.
Ausnahmen:
Falls pContent null ist, wird ein leerer Knoten erzeugt (und er erhält keine Verbindung zu pLeft/pRight).
Falls pLeft und/oder pRight null sind, wird stattdessen ein neuer leerer Knoten als Nachfolger erzeugt.

`boolean isEmpty()`
Gibt true zurück, falls dieser Knoten keinen Inhalt enthält (content = null).
Gibt false zurück, falls er einen Inhalt (und damit auch einen linken und rechten Nachfolger) enthält.

`void setContent(ContentType pContent)`
Falls dieser Knoten leer ist, wird „pContent“ als Inhalt gesetzt.
In diesem Fall werden zwei leere Knoten als linker / rechter Nachfolger erzeugt
(wie beim Konstruktor BinaryTree(ContentType pContent))
Falls der Knoten nicht leer ist, wird der aktuelle Inhalt durch „pContent“ ersetzt.

`ContentType getContent()`
Gibt das Inhalts-Objekt dieses Knotens zurück (null, falls der Knoten leer ist).

`void setLeftTree(BinaryTree pTree)`
Setzt den Binärbaum pTree als linken Teilbaum dieses Knotens (d.h. die Referenz left).
Falls bereits ein linker Teilbaum existiert (und keine weitere Referenz auf ihn zeigt), wird dieser gelöscht.

`void setRightTree(BinaryTree pTree)`
Setzt den Binärbaum pTree als rechten Teilbaum dieses Knotens (d.h. die Referenz right).
Falls bereits ein rechter Teilbaum existiert (und keine weitere Referenz auf ihn zeigt), wird dieser gelöscht.

`BinaryTree getLeftTree()`
Gibt eine Referenz auf den linken Nachfolger dieses Knotens (d.h. den linken Teilbaum) zurück
(null falls der Knoten leer ist).

`BinaryTree getRightTree()`
Gibt eine Referenz auf den rechten Nachfolger dieses Knotens (d.h. den rechten Teilbaum) zurück
(null falls der Knoten leer ist).

Aufgabe 1

a) Lies die Beschreibung des Konstruktors BinaryTree().
Er erzeugt einen „leeren Knoten“. Wie sieht dieser leere Knoten in einem Objektdiagramm aus?

b) Lies die Beschreibung der Methode setContent(…). Wie sieht ein zuvor leerer Knoten aus, nachdem man mit setContent einen Inhalt (z.B. einen String) eingefügt hat?

Aufgabe 2

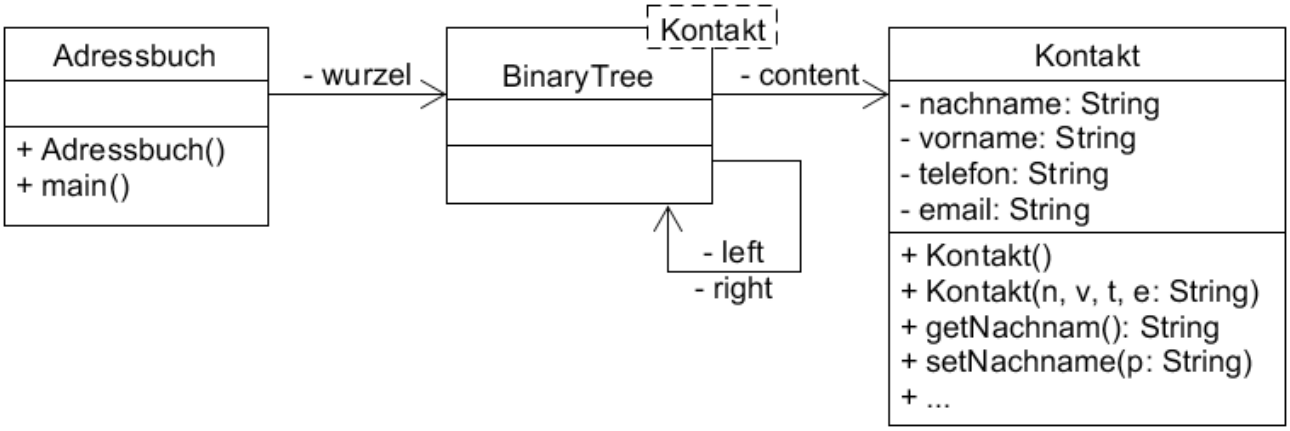
a) Schneide die Knoten- und String-Objekte auf Seite 5 mit einer Schere aus.
Lege sie auf einem leeren Blatt Papier zu einem Objektdiagramm, das die Situation am Ende der Methode füllen() darstellt. Zeichne die Referenzen zwischen den Objekten mit Bleistift.
Die Referenzen aktuell und neu sind lokal, d.h. sie gehören keinem Objekt.
Zeichne sie als Pfeile auf die Objekte, auf die sie am Ende der Methode zeigen.

b) Erläutere, wozu die Referenzen wurzel, aktuell und neu benutzt werden.
Was würde am Ende der Methode füllen() passieren, wenn wurzel auch lokal deklariert wäre?

```
01  public class Beispiel
02  {
03     BinaryTree<String> wurzel;
04
05     public Beispiel()
06     {
07        wurzel = new BinaryTree();
08     }
09
10     public void füllen()
11     {
12        BinaryTree<String> aktuell, neu;
13
14        wurzel.setContent("Müller");
15        neu = new BinaryTree("Günther");
16        wurzel.setLeftTree(neu);
17        neu = new BinaryTree("Peters");
18        wurzel.setRightTree(neu);
19
20        aktuell = wurzel.getLeftTree();
21        neu = new BinaryTree("Albrecht");
22        aktuell.setLeftTree(neu);
23
24        aktuell = wurzel.getRightTree();
25        neu = new BinaryTree("Neumann");
26        aktuell.setLeftTree(neu);
27     }
28  }
```

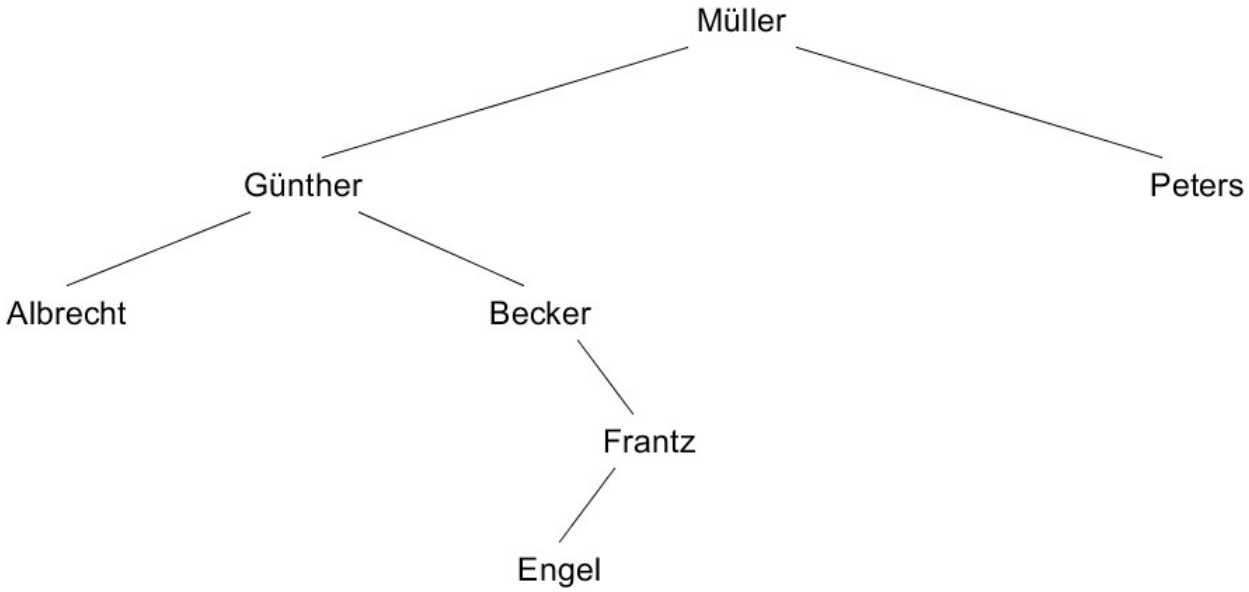
Aufgabe 3 – Binärbaum erzeugen

In der Aufgabe soll ein Binärbaum aufgebaut werden, der Kontakte enthält:



Verwende die ausgeteilte BlueJ-Vorlage. Implementiere die **main-Methode**.
Sie soll einen Binärbaum mit der folgenden Struktur aufbauen.

Hinweis: Für die Aufgabe sind nur die Nachnamen relevant, die restlichen Daten können mit leeren Strings ("") belegt werden.



Klasse BinaryTree