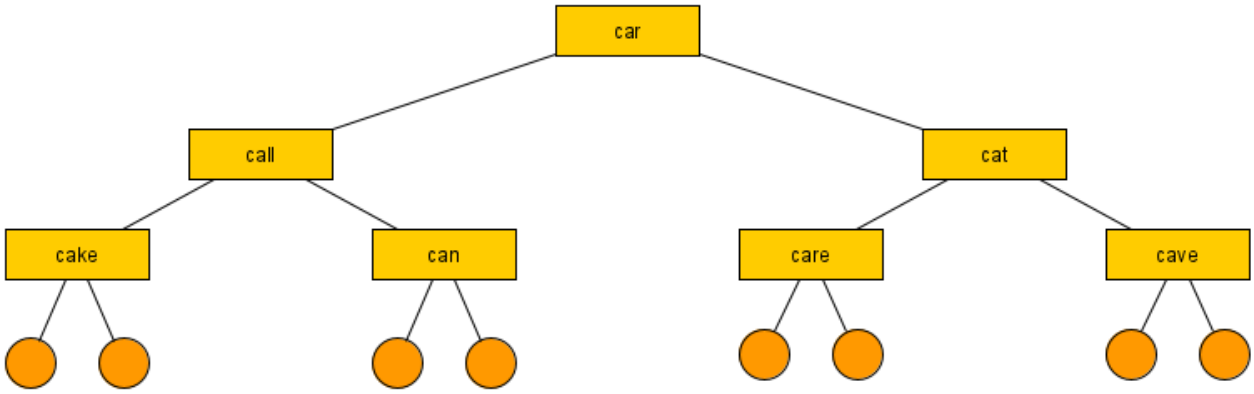
2.2 Einfügen und Suchen im BAUM

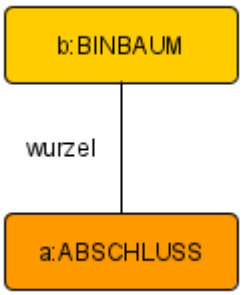
Wie wir gesehen haben, ist das effizientere Suchen eine der Hauptmotivationen für die Baumstruktur. Um in einem Baum suchen zu können, müssen wir jedoch zuerst einen BAUM aufbauen. Analog zur LISTE überlegen wir uns, dass eine rekursive Arbeitsweise sinnvoll ist. Betrachten wir dafür nochmal die Baumstruktur.



In gewisser Weise gilt bei beiden Methoden, dass wir so lange durch den BAUM springen, bis wir die passende Stelle gefunden haben. Dort muss dann je nachdem welche Methode aufgerufen wurde anders gehandelt werden.

Das Einfügen im BAUM

Aus dem Klassendiagramm wissen wir, dass jeder KNOTEN seine Nachfolger kennt. Die Blätter des BAUMes haben ABSCHLUSS Elemente als Nachfolger.
Wir müssen uns immer fragen, was passiert am Anfang, mit einem neuen leeren BAUM. Im leeren BAUM haben wir folgende Situation:



Direkt als Wurzel kommt der ABSCHLUSS.
Wir verwenden die Idee unserer LISTE und geben das neu einzufügende BAUMELEMENT an die Wurzel weiter. Wir merken, dass der BINBAUM b vom Nachfolger einen neuen Nachfolger bekommen muss. Das bedeutet, dass unsere Methode einen Rückgabewert braucht um mit dem Vorgänger kommunizieren zu können (analog zur LISTE).
Der ABSCHLUSS kann dann dafür sorgen, dass an der passenden Stelle eingefügt wird, indem er das Element als neuen Nachfolger an den BINBAUM b weitergibt.

Wir wissen nun also, dass wir sowohl in jedem BAUMELEMENT als auch im BINBAUM die Methode Einfügen mit Rückgabetyp BAUMELEMENT und Parameter DATEN brauchen.

Wir überlegen uns das als Struktogramme:

In BINBAUM:

```
void Einfuegen(DATEN d)
    wurzel = wurzel.Einfuegen(d)
```

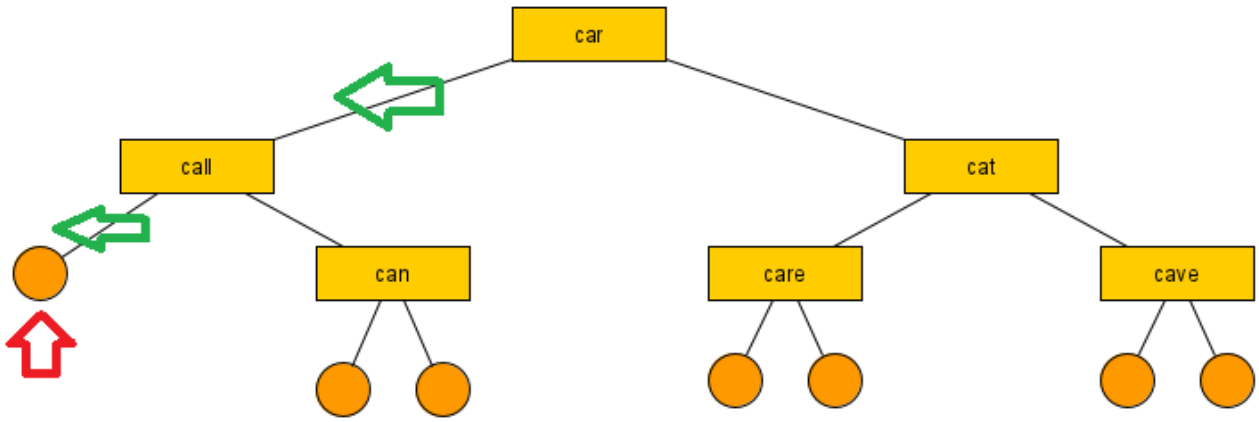
Der BINBAUM gibt die DATEN weiter an die Wurzel und übernimmt als Wurzel den Eintrag, den die Wurzel als Rückgabe gibt.

In ABSCHLUSS:

```
BAUMELEMENT Einfuegen(DATEN d)
    BAUMELEMENT r = new KNOTEN(d)
    r.LinksNachfolgerSetzen(this)
    r.RechtsNachfolgerSetzen(this)
    return r
```

Der ABSCHLUSS erstellt einen neuen KNOTEN. Er setzt sich selber als linken und rechten Nachfolger (Achtung: hier immer neue ABSCHLUSS Elemente zu erschaffen ist überflüssig und braucht ggf. sehr viel Speicher bei langen Listen!)
Am Schluss gibt er das neue Element an den Vorgänger zurück.

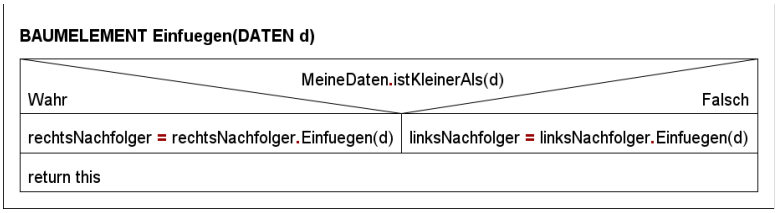
Aus dem einfachen Fall des leeren BINBAUM haben wir also die Handlungsweisen von BINBAUM und ABSCHLUSS abgeleitet. Jetzt fehlt uns nur noch die Handlung der inneren KNOTEN. Dabei stellen wir fest, dass diese eigentlich nie selber KNOTEN einfügen. Sie müssen nur dafür suchen, dass die DATEN an die passende Stelle weitergeleitet werden. Nehmen wir mal an, in unserem Standardbeispiel würde „cake“ noch fehlen. Was müsste beim Einfügen passieren?



Wir in der Zeichnung illustriert, müssten vom KNOTEN „car“ die DATEN weitergegeben werden an den KNOTEN „call“. Die Entscheidung hängt davon ab, dass „cake“ lexikographisch kleiner ist, als „car“ (d.h. im Lexikon kommt „cake“ vor „car“ da alphabetisch „k“ vor „r“ kommt). Danach geht es mit „call“ weiter, der das gleiche macht und somit entscheidet, dass „cake“ links von „call“ kommen muss. Wir sehen an dem „der-das-gleiche-macht“ die Rekursion in der Methode.

Überlegen wir uns das als Struktogramm:

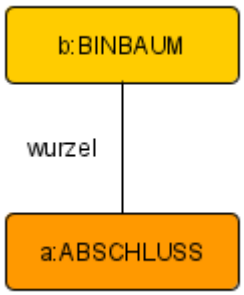
In KNOTEN:



Zuerst überprüft der KNOTEN, ob seine Daten kleiner sind. Wenn ja dann muss rechts eingefügt werden sonst links. Am Ende gibt der KNOTEN immer sich selber an seinen Vorgänger zurück, da sich seine Position nicht verändert.

Das Suchen im BAUM

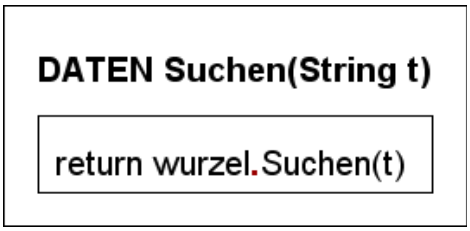
Suchen im BAUM geht nun wieder rekursiv. Wir beginnen wieder mit unserem leeren BINBAUM und überlegen, was die Aufgabe des ABSCHLUSS ist.



Wir überlegen, wie die Methode gestaltet sein muss. Wir erwarten beim Suchen selbstverständlich, dass das gesuchte Element ausgegeben wird. Das heißt wir brauchen auf jeden Fall einen Rückgabetyp. Man kann sowohl für DATEN als auch für BAUMELEMENT Gründe finden. In unserem Beispiel geben wir die DATEN aus, da wir beim Suchen in einem Wörterbuch vermutlich die Übersetzung, also die DATEN brauchen. Parameter sollte ein „Suchschlüssel“ sei, nach dem wir suchen. Es bietet sich ein Text also String als Datentyp an.

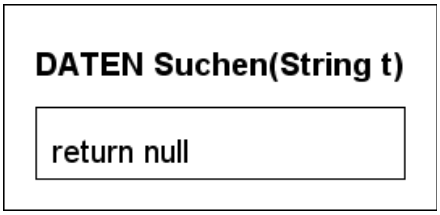
Der Ablauf der Methoden gestaltet sich in den Struktogrammen dann wie folgt:

In BINBAUM:



Der BINBAUM gibt den Befehl einfach an seine Wurzel weiter. Sie sollte ihm das DATEN Objekt des passenden KNOTEN geben.

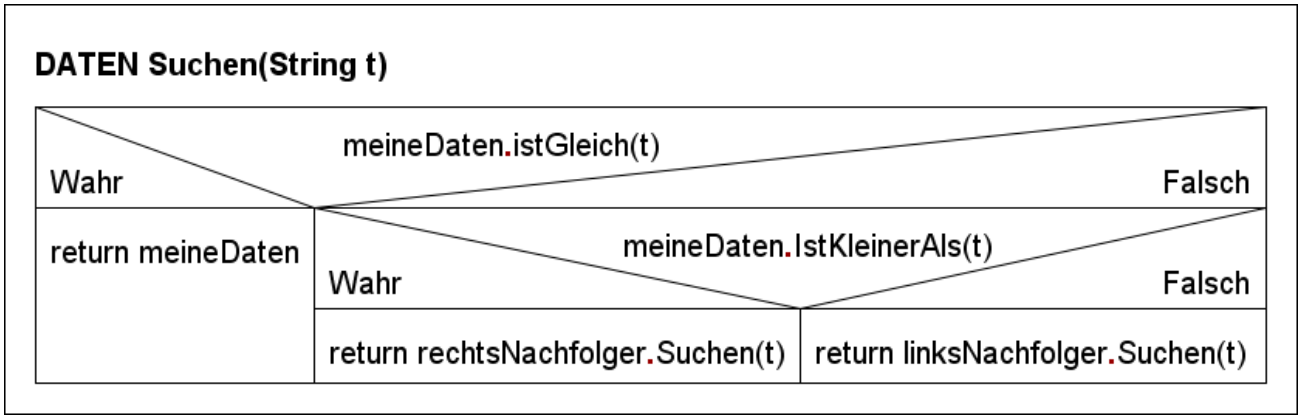
In ABSCHLUSS:



Der ABSCHLUSS wird nur dann vom Suchen erreicht, wenn das Element nicht im BAUM vorhanden ist. Daher gibt er „null“ aus.

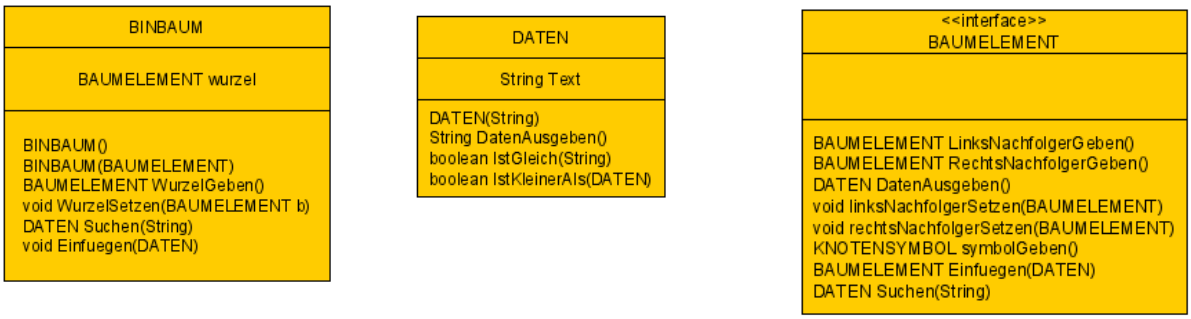
(Anmerkung für Spezialisten: Es wäre eleganter eine Exception zu werfen in den Methoden der BAUMELEMENTe. Diese würde dann zu einem try-catch in BINBAUM zwingen, das geht jedoch über den Stoff der Q11 hinaus...)

Es fehlt uns nur noch die Betrachtung des KNOTEN. Hier stellen wir fest, dass es quasi drei Fälle gibt. Entweder halte ich die DATEN, dann gebe ich sie aus, oder ich muss dafür sorgen, dass rechts oder links weitergesucht wird. Das sieht im Struktogramm übersichtlich dann so aus:



Klassendiagramme BAUMELEMENT, BINBAUM und DATEN

Zum Überblick hier nochmal die fertigen Klassendiagramme:



Wie wir bei Suchen und Einfügen gesehen haben, brauchen unsere DATEN unbedingt Methoden, die uns ermöglichen diese mit anderen DATEN auf Gleichheit und Ordnung zu untersuchen. Damit ist es nötig den DATEN zwei Methoden

```
boolean istGleich(String vergleich)
```

und

```
boolean istKleinerAls(DATEN vergleich)
```

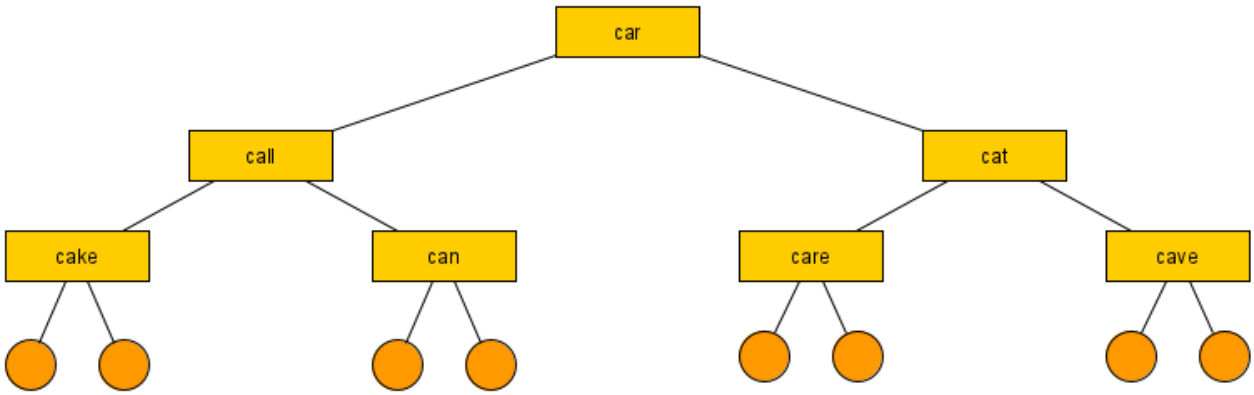
hinzuzufügen. Dabei sollte `istKleinerAls` nur dann true ausgeben, wenn die DATEN echt kleiner sind.

Aufgaben:

1) Kopiere die Vorlage aus dem Klassenordner (Swing_BinBaum_02).

2) Baue in die Klassen BAUMELEMENT, KNOTEN und ABSCHLUSS die neuen Methoden zum Suchen und Einfügen ein.

3) Teste deinen BAUM indem du ihm vom leeren BAUM langsam die Elemente des Beispiels hinzufügst. Teste, wie der BAUM aussieht, wenn du die Reihenfolge der Befehle tauschst. Woran liegt das Phänomen, das du feststellst? In welcher Reihenfolge musst du die KNOTEN einfügen, um den unten gegebenen BAUM zu erhalten?



4) Schreibe eine rekursive Methode, die die Tiefe eines BAUM bestimmt und zurückgibt.

5) **(Achtung sehr schwer und umfangreich!!!)** Schreibe eine weitere Methode, die Überprüft, ob der BAUM ideale Höhe hat und falls nicht den BAUM neu aufbaut, so dass er ideal gebaut ist.