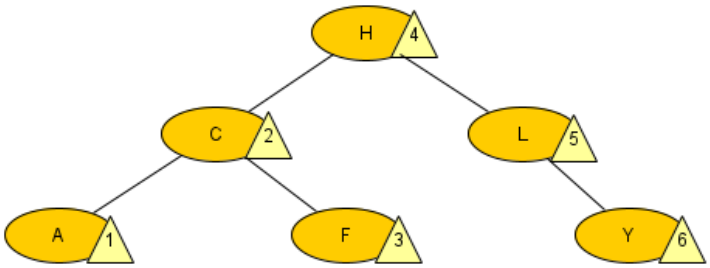
2.3 Ausgeben des Inhalts eines Baumes - Baumdurchläufe

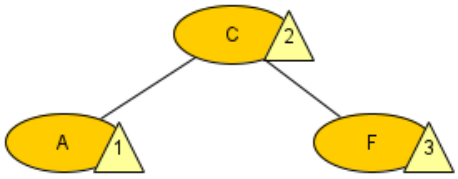
Im Allgemeinen, reicht es nicht, dass man in unserem BAUM nur suchen kann. Oft ist es nötig sich den ganzen Bauminhalt ausgeben zu lassen. Für den Durchlauf durch den ganzen BAUM gibt es drei verschiedene Ansätze, je nachdem in welcher Reihenfolge man die KNOTEN ausgibt. Jede der Möglichkeiten hat unterschiedliche Anwendungsfelder.

2.3.1 Der InOrder Durchlauf

Nehmen wir als Beispiel wieder unser Wörterbuch. Im Wörterbuch sucht man oft nicht nur nach einem speziellen Wort, sondern muss bei der Darstellung des Wörterbuchs die Wörter in der lexikographischen Reihenfolge ausgeben können.
Überlegen wir uns anhand eines Beispiels die „Ausgabereihenfolge“ der KNOTEN, kommen wir zu folgendem Ablauf:



Um zu verstehen, wie die KNOTEN besucht werden, reduzieren wir den BAUM auf den linken unteren Teilbaum:



Wir sehen, dass von jedem KNOTEN aus zuerst der linke Nachfolger, dann der KNOTEN selber und zum Schluss der rechte Nachfolger ausgegeben wird.
Das Prinzip lässt sich auch auf den größeren BAUM erweitern. Solange linke Nachfolger vor dem KNOTEN selber genannt werden und die rechten Nachfolger danach kommen, bleibt die Reihenfolge erhalten.

Daraus ergibt sich folgender Algorithmus für jeden KNOTEN:

1. Gib an meinen linken Nachbarn weiter
2. Gib meine Informationen aus
3. Gib an meinen rechten Nachbarn weiter

Das müssen wir nun wieder in unsere Klassen ABSCHLUSS, KNOTEN und BINBAUM übersetzen. Die Ausgabe soll uns dabei über die Konsole genügen. Die Klasse DATEN hat hierfür eine Methode DatenAufKonsole parat, die genau das tut.
Schauen wir uns den ABSCHLUSS und BINBAUM zuerst an, da diese relativ wenig zu tun haben.

In BINBAUM:

```
void InOrder()
    wurzel.InOrder()
```

In ABSCHLUSS:

```
void InOrder()
    //Tut nichts
```

Der BINBAUM gibt den Befehl nur an seine Wurzel weiter.
Der ABSCHLUSS ist nur dafür da, dass wir nicht Probleme mit einer „null“ Referenz bekommen und gibt somit gar nichts aus.

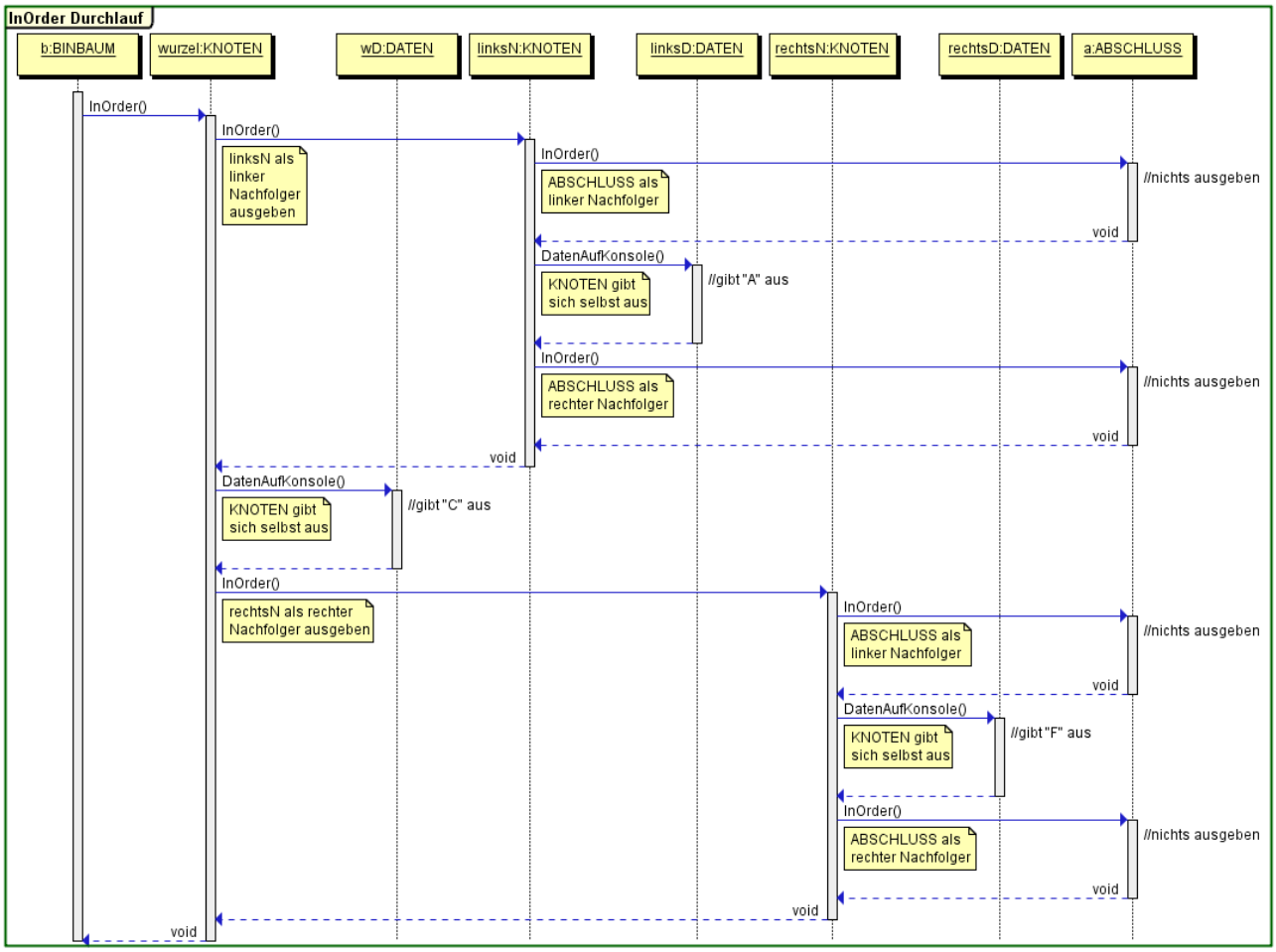
Der KNOTEN hat etwas mehr zu tun:

In KNOTEN:

```
void InOrder()
    LinksNachfolger.InOrder()
    MeineDaten.DatenAufKonsole()
    RechtsNachfolger.InOrder()
```

Der KNOTEN weiß, dass im lexikographischen Sinne zuerst Elemente in seinem linken Teilbaum ausgegeben werden müssen, also gibt er den Befehl zuerst nach links. Wenn der Befehl zu ihm zurückkommt, dann gibt er seinen eigenen Datensatz aus. Anschließend gibt er an seinen rechten Nachbarn weiter.

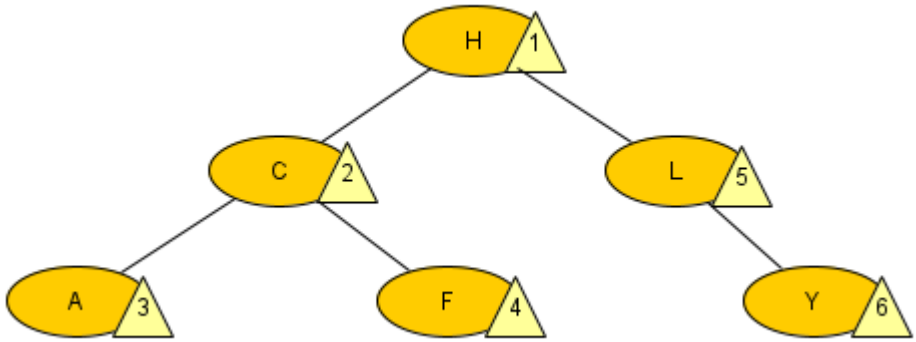
Interessant ist dabei, dass die Rekursion der Methode verhindert, dass der KNOTEN zu früh seinen Wert ausgibt. Er muss ja warten, biss der nächste KNOTEN den er als linken Nachfolger aufgerufen hat ihm ein „void“ zurückgibt. Vorher wird er nicht aktiv!
Wir „veranschaulichen“ uns den Ablauf in einem (zugegebenermaßen sehr langen) Sequenzdiagramm:



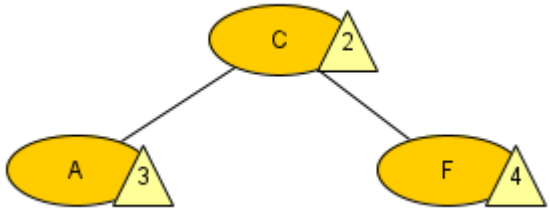
Es ist zwar etwas lang, veranschaulicht jedoch trotzdem den Ablauf gut und zeigt vor allen das „Warten“ auf den „return“ des Nachfolgers.

2.3.2 Der PreOrder Durchlauf

Wir haben bereits im Kapitel mit dem Einfügen in einer Übung festgestellt, dass beim Einfügen von KNOTEN meist keine schönen Bäume herauskommen. In vielen Anwendungen soll ein BAUM über eine geeignete Ausgabe „kopiert“ werden. Dafür müssen wir jedoch die KNOTEN in einer gewissen Reihenfolge Einfügen, damit wieder der gleiche BAUM rauskommt. Schauen wir uns ein Beispiel an, stellen wir fest, dass wir die KNOTEN in folgender Reihenfolge brauchen, damit wir beim Einfügen in dieser Reihenfolge den gleichen BAUM bekommen:



Wir vereinfachen das ganze wieder zu unserem unteren linken Teilbaum und stellen fest:



Diesmal muss der KNOTEN sich selbst zuerst nennen, da er eingefügt werden muss, bevor man seine Nachfolger einfügen kann. Deswegen kommt er selber zuerst und dann die beiden Nachfolger links und rechts.

Daraus ergibt sich folgender Algorithmus für jeden KNOTEN:

1. Gib meine Informationen aus
2. Gib an meinen linken Nachbarn weiter
3. Gib an meinen rechten Nachbarn weiter

In den Klassen BINBAUM und ABSCHLUSS tut sich nicht viel im Vergleich zum InOrder Durchlauf. Sie machen das exakt gleiche:

In BINBAUM:

```
void PreOrder()
    wurzel.PreOrder()
```

In ABSCHLUSS:

```
void PreOrder()
    //Tut nichts
```

Wie beim InOrder-Durchlauf ist hat der BINBAUM nur die Aufgabe die Aufgabe weiter zu geben und der ABSCHLUSS die Aufgabe nicht zu tun und einen Fehler wegen „null“ Referenz zu umgehen.

Erst beim KNOTEN wird es wieder etwas spannender, er setzt quasi wörtlich unsere Überlegung von oben um:

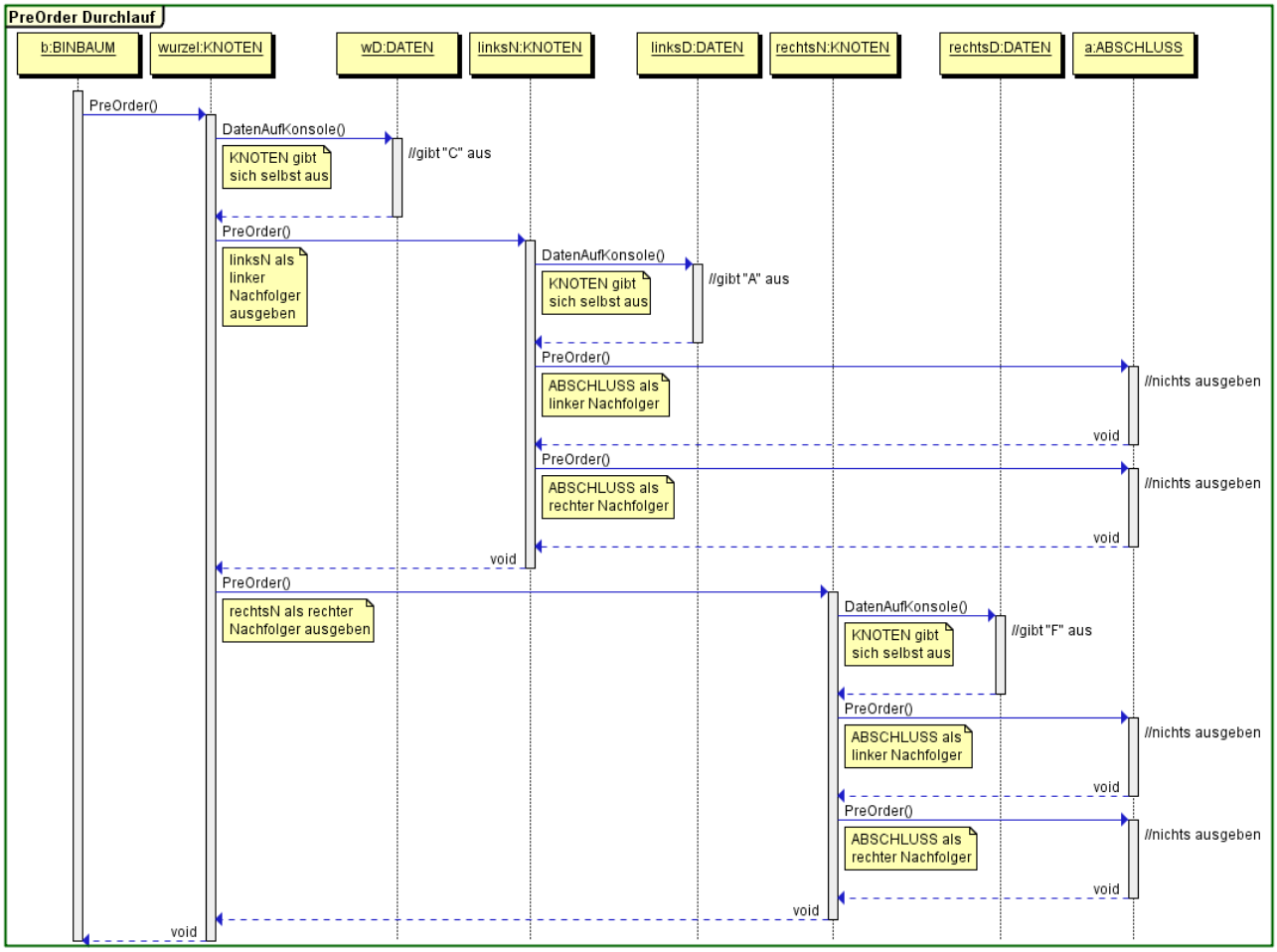
In KNOTEN:

```
void PreOrder()

    MeineDaten.DatenAufKonsole()
    LinksNachfolger.PreOrder()
    RechtsNachfolger.PreOrder()
```

Der KNOTEN gibt diesmal sich selbst zuerst aus und dann seine linken und rechten Nachfolger. Er wartet automatisch durch die Rekursion wieder so lang, bis die linken fertig sind, bevor er mit den rechten weiter macht.

Der Vollständigkeit halber schauen wir uns auch hier ein (umfangreiches) Sequenzdiagramm an:



Wir sehen wieder das „Warten“ auf die Antwort („void“) der anderen Nachfolger, bevor es auf die rechte Seite geht.

2.3.3 Anmerkungen:

Bei allen Durchläufen könnte man auch immer rechts vor links abhandeln. Dann würde bei der „InOrder“-Variante die Einträge rückwärts ausgegeben und bei der „PreOrder“ Variante würden auch die rechten Einträge vor den linken ausgegeben, was hier bei der Anwendung „Kopieren-des-Baumes“ keinen Unterschied machen würde.

Aufgaben:

1) Betrachte die gegebenen Bäume. Kannst du erklären, welche KNOTEN nacheinander besucht werden und welche ausgegeben werden? Veranschauliche den Weg auf einem Blatt Papier und schreibe daneben die Ausgabe.

2) Kopiere die Vorlage aus dem Klassenordner (Swing_BinBaum_03). Wenn du nicht mit der Vorlage weiter arbeitest, gehe sicher, dass deine DATEN Klasse eine Methode hat, die den Inhalt der DATEN auf die Konsole ausgibt.

3) Ergänze in den Klassen BAUMELEMENT, ABSCHLUSS, KNOTEN und BINBAUM die Methoden `InOrder()` und `PreOrder()` und implementiere sie mithilfe der überlegten Struktogramme. Teste die Durchläufe mit der Klasse „TESTBAUM“ oder einem beliebigen eigenen Baum.

4) Es gibt noch eine weitere Möglichkeit durch den BAUM zu gehen. Der Anwendungsfall ist folgender: Stell dir vor die KNOTEN im BAUM sind Aufgaben. Die Blätter sind Aufgaben, die sofort erledigt werden können und alle inneren KNOTEN brauchen dass alle Nachfolger-Aufgaben erledigt sind, bevor sie erledigt werden können. Man nennt diese Art des Durchlaufs PostOrder-Durchlauf.

> **a)** Überlege, anhand eines Beispieles, in welcher Reihenfolge ein BAUM besucht werden müsste, um die KNOTEN in diesem Sinne auszugeben.
>
> **b)** Überlege dir das Struktogramm für die Methode `PostOrder()` in der Klasse KNOTEN und setze es in deinem BAUM um.
>
> **c)** Versuche anhand eines rudimentären Sequenzdiagrammes zu erklären, wie die Ausgabe des Standardbeispieles (siehe oben) ablaufen würde.

5) Achtung schwer! Die Daten auf die Konsole auszugeben ist eigentlich nicht sinnvoll. Um zum Beispiel einen BAUM anhand einer PreOrder Ausgabe wieder aufzubauen sollte die Reihenfolge in einem günstigen Datentyp gespeichert werden.
Überlege dir eine neue PreOrder Methode (Tipp: mit ggf. Rückgabewert und Parameter), die es ermöglicht die Reihenfolge z.B. in einem String zu speichern und dann auch wieder aus diesem String wiederherzustellen.(z.B. Der Standardbaum von oben wird als „A C H“ ausgegeben und kann aus dem String „A C H“ wieder hergestellt werden.)
HINWEIS: (Für das Wiederherstellen) Suche im Internet nach einer günstigen Art, aus einem String Teile auszulesen und zu entfernen.