Die insert-Methode

Jetzt wird es schwierig. Wir wollen die insert-Methode implementieren. Dazu überlegen wir noch einmal ganz genau, was insert eigentlich leisten soll.
Wenn der Baum leer ist, geht insert ganz einfach: erzeuge einen leeren Knoten und schreibe dann den übergebenen Wert **element** in die inhalt-Komponente dieses Knotens. Der Quelltext, der dieses leisten würde, sähe ungefähr so aus:

```
knoten = new Knoten(element);
```

Nur ist uns damit nicht viel geholfen. Nach dem ersten insert-Aufruf ist der Baum leider nicht mehr leer. Was muss unser Algorithmus dann machen? Spielen wir das Ganze doch einfach mal durch. In der Wurzel befindet sich die 50, und die 30 soll jetzt eingefügt werden.
Zunächst muss nachgefragt werden, ob die Wurzel leer ist. Denn dann könnte ja der eben beschriebene einfache Teilalgorithmus angewandt werden. Falls die Wurzel nicht leer ist, muss der neue Wert (also die 30) mit dem Wert der Wurzel verglichen werden (mit der 50).
Wenn der neue Wert kleiner ist als der Wurzelwert, so ist im linken Teilbaum weiterzumachen. Andernfalls im rechten. Das war's. Moment mal, das kann doch gar nicht funktionieren, mag jetzt vielleicht jemand einwenden Bei ***rekursiven Algorithmen*** ist das nun mal so. Sie sehen total einfach aus, funktionieren aber – wenn man sie richtig programmiert. Andernfalls kann natürlich der ganze Rechner abstürzen.

```
if (wurzel == null)
        Einfacher Algorithmus (siehe oben)
else if (element < wurzel.wert)
        wurzel.links.insert(element);
    else wurzel.rechts.insert(element);
```

Achtung: das ist noch nicht der lauffähige Quelltext, sondern nur der Algorithmus. Aber ist er nicht bestechend einfach? Das ist das Schöne an der rekursiven Programmierung.

Einbettung des rekursiven insert-Algorithmus in die Klassenimplementation

Was für eine komplizierte Überschrift! Aber so ist es. Formulieren wir den rekursiven insert-Algorithmus noch einmal in JAVA, diesmal im richtigen Quelltext. Wir müssen diesen Algorithmus jetzt in die Implementation der Methode ***insert*** einbauen.

```
public void insert(int element) {
    if (wurzel == null)
        wurzel = new Knoten(element);
    else if (element < wurzel.wert)
        wurzel.links.insert(element);
    else
        wurzel.rechts.insert(element);
}
```

Rekursion bei der insert-Methode

Viele Schüler und Studenten haben Probleme, zu verstehen, warum der insert-Algorithmus funktioniert, obwohl er so einfach aussieht.
Betrachten wir dazu einmal die rechts abgedruckte Folie. Das Element mit der Nummer 42 soll in den bestehenden Baum eingefügt werden. Dazu wird die insert-Methode der Klasse BinTree aufgerufen, und die 42 wird als Parameter an diese Methode übergeben.
Auf der Folie wird nun gezeigt, wie nacheinander vier verschiedene Ebenen der internen insert-Methode aufgerufen werden. Der Trick bei der ganzen Sache ist der, wenn die erste insert-Ebene wieder insert aufruft, so landet der Algorithmus in der zweiten insert-Ebene. Damit ist aber die erste insert-Ebene noch nicht abgearbeitet. Wenn nämlich die zweite insert-Ebene verlassen wird, macht der Algorithmus wieder in der ersten insert-Ebene weiter, und zwar an der Stelle, an der er dort aufgehört hatte, um in die zweite Ebene zu verzweigen.
Da aber nach dem rekursiven Aufruf von insert keine Codezeile mehr kommt, ist die Methode mit dem Rücksprung in die höhere Ebene praktisch beendet.