Java Tutorial

Part 1 - Elementare Grundlagen

1. Zahlensysteme

Elementarer Bestandteil der Informatik und damit auch der Java Entwicklung ist die Kenntnis über verschiedene Zahlensysteme. Im Speziellen wäre das das binäre (lat. „bini“, für „zwei“) Zahlensystem, dass einen Zeichenvorrat von zwei (0, 1) besitzt.

Viele Leute, die sich nicht näher mit der Informatik befassen, verstehen unter dem Programmieren ein schreiben von ewigen Nullen und Einsen in einer unverständlichen Reihenfolge. Mehr oder weniger macht man das auch, nur halt eben nicht direkt, unter anderem ist es deshalb von Vorteil über das binäre Zahlensystem bescheid zu wissen.

1.1 Dezimalsystem

Das Dezimalsystem (lat. „decem“, für „zehn“) ist das gebräuchliche Zahlensystem der Menschheit, es hat im Gegensatz zu seinem Binären Verwandten einen Zeichenvorrat von zehn (0, 1, 2, …, 9). Jedem Kind wird heute dieses System bereits in der Grundschule beigebracht und zur Basis für die spätere Mathematik vorbereitet. Vielen sind einige Hintergründe aber nur indirekt bekannt, sie wissen über diese nicht bescheid, nutzen sie aber unbewusst tag für tag.

1.1.1 Aufbau der Zahlen

Jede Zahl hat einen gewissen Aufbau, anhand der Dezimalzahlen möchte ich dies zunächst klar machen um einen späteren Transfer auf das Binärsystem zu vereinfachen.

Nehmen wir als Beispiel die Zahl 112. In der Grundschule hat man vielleicht eine solche Zahl auch mal wie folgt notiert:

Hunderter	Zehner	Einser
1	1	2

Das System ist eigentlich relativ einfach und für jeden Grundschüler ohne weiteres verständlich, nun möchten wir das System etwas erweitern. Aus der Mittelstufe sollten die 10er Potenzen bekannt sein beziehungsweise verinnerlicht sein. Die Bezeichnungen „Hunderter“, „Zehner“, „Einser“ usw. ersetzen wir nun durch das Mathematische Äquivalent dazu.

10^2	10^1	10^0
1	1	2

Ein System ist nun erkennbar, denn $112 = 1*10^2 + 1*10^1 + 2*10^0$. Dieses Wissen verwenden wir in abgewandelter Form bei dem binären Zahlensystem.

1.2 Binärsystem

Wie nun bereits bekannt haben wir bei dem Binärsystem einen ausschließlichen Zeichenvorrat von zwei, bestehend aus der Null und der Eins, dennoch ist es möglich Zahlen darzustellen, als Beispiel nehme ich im folgenden wieder die 112.

1.2.1 Aufbau der Zahlen

Sie sollten nun erkennen, dass es ein Problem gibt: Wie stellt man eine Zahl größer 1 mit dem System dar. Bis zur 1 ist das ganze ja relativ trivial.

Dezimal	Binär
0	0
1	1
2	?

Bereits bei der zwei ist unser Zeichenvorrat erschöpft, denn mehr als die eins und die null haben wir nicht, der einzige Ausweg ist das hinzufügen einer weiteren Stelle (also ein Platz für ein weiteres Bit, denn jede Stelle stellt ein Bit dar), wie wir es von den Dezimalzahlen auch kennen. Dort ist nach der neun Schluss, man fügt eine neue Stelle hinzu, beginnt dort bei der auf der neuen Stelle und „setzt“ die erste Stelle auf null.

Theoretisch wäre die 2 im Dezimalsystem auch als 02 oder 002 usw. zu notieren, nur macht das wenig Sinn, die Null hat nunmal vorangestellt keinen Wert. Sie fällt deshalb weg. Ähnliches betrachten wir beim Binärsystem. 00000 bleibt 0, 0000001 bleibt 1. Man kann also altbekanntes auch hier anwenden.

Nun zur eigentlichen Problemlösung: Für das Dezimalsystem haben wir die Zehnerpotenzen benutzt, für das Binärsystem nutzen wir die Zweierpotenzen. Zunächst ein Beispiel an der zwei.

2^2	2^1	2^0
0	1	0

Aus der Tabelle wir nun ersichtlich $2_{10} = 010_2 = 10_2$. D.h. $2_2 = 10_2$
Die tiefgestellte zwei bzw. zehn steht für das System. Zwei für das Binärsystem, 10 für das Dezimalsystem. Das ganze wir für die 112 nun etwas aufwändiger, bleibt aber nach wie vor machbar.

Die Gleichung lautet also wie folgt:
$1*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 0*2^0 = 112$
So rechnen wir vorzugsweise aber von Binär -> Dezimal, andersrum ist das ganze mit großem Aufwand und hoher Fehleranfälligkeit verbunden.

Hier nun eine Tabelle zur Lösung dieses Problems, man geht einfach Schritt für Schritt durch die Tabelle:

Zur Erklärung der Tabelle:

x Spalte
In die erste Zeile der x Spalte kommt die Zahl, die man umrechnen möchte. In die weiteren Zeilen der x Spalte folgt nun der Inhalt der vorigen $div_2(x)$ Spalte.

$div_2(x)$ Spalte
$div_2(x)$ steht für die Vollzahldivision (= Integerdivision) der Zahl x mit 2. x ist hierbei eine Variable, die jeweils durch den Wert der x Spalte in der aktuellen Zeile ersetzt wird. Es wird bei dieser Division durch den Divisor 2 geteilt und abgerundet, der Rest, also die „Kommastellen“ lässt man wegfallen.

$mod_2(x)$
$mod_2(x)$ steht für die Restdivision (= Modulo) der Zahl x mit 2. D.h. es wird das Überbleibsel der Vollzahldivision hier aufgeschrieben. Entweder man rechnet dieses separat aus oder man rechnet einfach den Wert der $div_2(x)$ Spalte mal 2 und zieht diesen Wert von x ab ($x - div_2(x) * 2$). In der ersten Zeile würde das wie folgt aussehen:
$112 - 56 * 2 = 0$

Um nun zur Binärzahl zu kommen, geht man die $mod_2(x)$ von unten nach oben rauf.
$\Rightarrow 1110000_2$

1.2.2 Operationen
Auch auf der binären Zahlenebene lassen sich die gewohnten Operationen Addition, Subtraktion, Multiplikation, Division durchführen, auf diese möchte ich aber nicht näher eingehen. Ich möchte nur auf ein weiteres Problem bei der Darstellung im Rechner eingehen.

1.2.3 Negative Zahlen
Siehe dazu: http://de.wikipedia.org/wiki/Zweierkomplement
Wichtig für das weitere Dokument zu wissen, dass ein Bit für das Vorzeichen reserviert ist (gilt nicht für unsigned Datentypen) und somit für den Zahlraum nicht direkt nutzbar ist.

1.2.4 Dezimalzahlen
Dezimalzahlen werden in der Informatik annäherungsweise dargestellt.
Siehe dazu: http://de.wikipedia.org/wiki/Gleitkommazahl

2. Warum wird das ganze nun benötigt?
Zunächst sollte klar sein, dass die Zahlen in einem Rechner binär dargestellt sind, daher auch die landläufige Meinung, Programmierer schreiben nur Einsen und Nullen. Da jede Zahl als eine Folge von verschiedenen Bits dargestellt wird, braucht man für jede Zahl eine gewisse Anzahl an Stellen. Die Anzahl an Stellen kann man durch folgende Funktion bestimmen:
$f(x) = \lceil((\log_2 x)+1)\rceil$
Warum ist das nun wichtig? Ganz einfach, betrachten wir den Datentyp Integer in Java, er wird mit 4 Byte dargestellt. 1 Byte sind 8 Bit, damit haben wir zur Darstellung eines Integers 32 Bit. Wir können also genau 2^{31} Zahlen darstellen. 2^{31}-1 ist dabei die größte darstellbare Zahl, -2^{31} die Kleinste. Addieren wir also zur Zahl 2^{31}-1 die Zahl 1, so überschreiten wir den Zahlbereich der Integers und haben damit einen Überlauf. Solche Überläufe werden zwar in unserem Fall von der JavaVM abgefangen und können somit keinen massiven Schaden anrichten, doch können sie nach wie vor ungewollte Fehlfunktionen des Programms auslösen. Bei Java fängt die Zahl beim Überlauf über Maximum wieder beim Minimum an. Überläuft man das Minimum, endet man wieder im Maximum. Man kann es sich also als Ring vorstellen, der unendlich in beide Richtungen überschreitbar ist. Man fängt entweder wieder von unten oder von oben an.