Rekursion - Teil I

Um Rekursion zu verstehen, muss man entweder einen kennen,
der sie versteht, oder sie schon verstanden haben.
(M. Freericks)

Der Begriff *Rekursion* ist in der Tat nicht ganz einfach zu verstehen. Auch die Definition des Informatik-Dudens lässt Fragen offen:

Definition der Rekursion

Definition eines Problems, einer Funktion oder eines Verfahrens durch sich selbst. Genauer: Das Ergebnis eines Verfahrens für die Eingabe x wird auf das Ergebnis des gleichen Verfahrens für die Eingabe y zurückgeführt.
Ouelle: Informatik-Duden

Gemeint ist damit folgendes:

Man spricht von Rekursion, wenn ein Problem dadurch gelöst wird, dass eine Methode aufgerufen wird, die sich selbst wieder selbst aufruft.

Es gibt auch mathematische Definitionen, die rekursiv definiert sind (vgl. Aufgabe 1).

Letztlich lässt sich Rekursion wohl am besten mit "*Selbstaufruf*" übersetzen.

Beispiel 1: Einfache Rekursion

```
/**
  * Einfache Rekursion
  */

public class Selbstaufruf {

  public static void methode(){
    System.out.println("Es war einmal ein Mann, der hatte vier Söhne ...");
    methode();          // Rekursion
  }

  public static void main(String[] args) {
     methode();
  }
}
```

Es ist klar, dass es wenig Sinn macht, dieses Programm auszuführen, denn hier hat man natürlich eine Endlosschleife programmiert! Trotzdem erkennt man, dass die Methode sich selbst aufruft. Es ist somit eine *rekursive* Methode.

Fast alle Programmiersprachen (außer Fortran, Cobol und Basic) unterstützen rekursive Methodenaufrufe. Es ist zwar grundsätzlich immer möglich, ein Problem, das rekursiv gelöst wird, auch iterativ (also mittels Schleifen) zu lösen, doch häufig ist der rekursive Ansatz klarer und einfacher verständlich und die iterative Version vollkommen undurchschaubar. Allerdings gibt es auch Nachteile bei der Lösung eines Problems mittels Rekursion, auf die wir im Folgenden noch eingehen werden.

Beispiel 2: Berechnung der Summe von n Zahlen

Iterative Lösung

```
/**
  * Summe von n Zahlen (iterativ)
  */

public class SummeIterativ {

  public static long summe(int n){
    long sum = 0;

    for (int i=0;i<=n;i++) {     // Iteration
      sum = sum + i;
    }
    return sum;
  }

  public static void main(String[] args) {

     System.out.println("Dieses Programm berechnet die Summe der Zahlen von 1 bis n.\n");
     int n = IOTools.readInt("Geben Sie eine Zahl n>=0 ein: ");

     if(n>=0){
       System.out.println("\nDie Summe der Zahlen von 0 bis "+n+" lautet: "+summe(n));
     }else{
       System.out.println("\nFalsche Eingabe!");
     }
  }
}
```

Rekursive Lösung

```
/**
  * Summe von n Zahlen (rekursiv)
  */

public class SummeRekursiv {

  public static long summe(int n){
    if (n==0){
      return 0;                  // Abbruch
    }else{
      return n + summe(n-1);     // Rekursion
    }
  }

  public static void main(String[] args) {

     System.out.println("Dieses Programm berechnet die Summe der Zahlen von 1 bis n.\n");
     int n = IOTools.readInt("Geben Sie eine Zahl n>=0 ein: ");

     if(n>=0){
       System.out.println("\nDie Summe der Zahlen von 0 bis "+n+" lautet: "+summe(n));
     }else{
       System.out.println("\nFalsche Eingabe!");
     }
  }
}
```

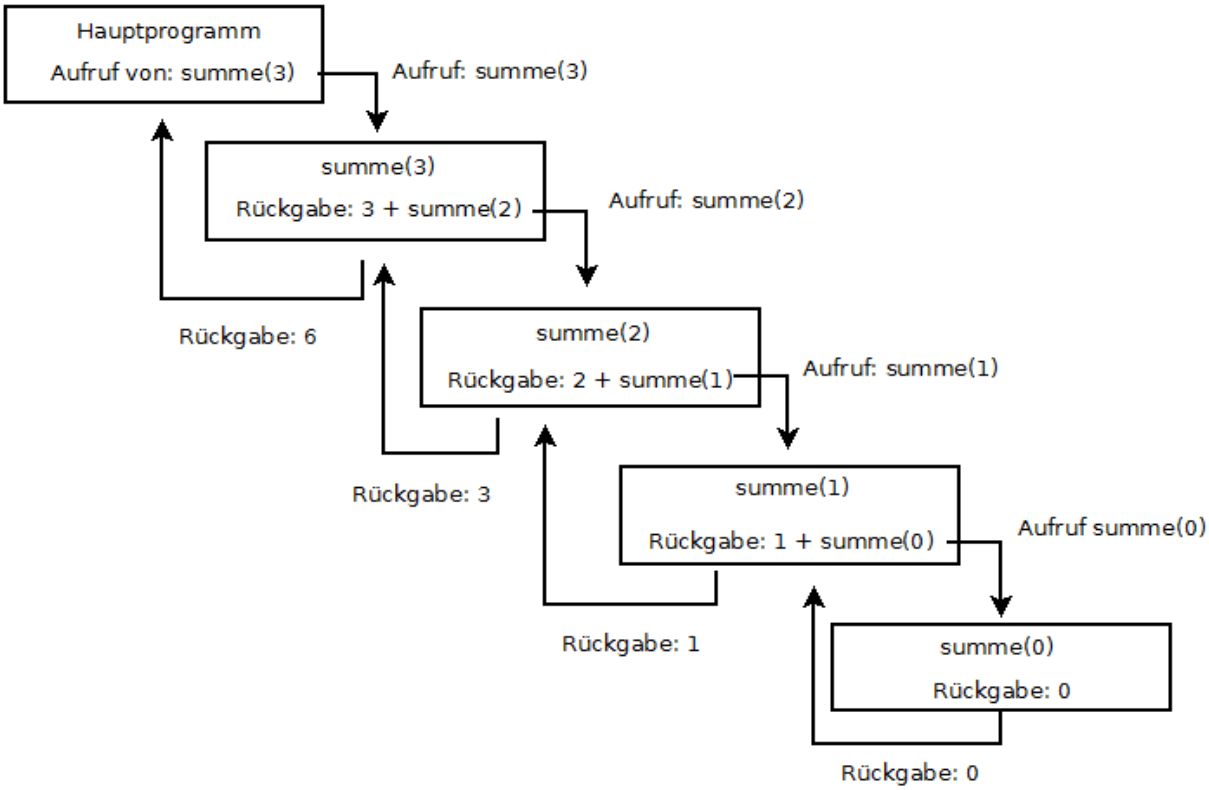
Das Beispiel zeigt, dass der rekursive Ansatz kürzer und klarer ist und zudem ohne Variablen auskommt!

Was passiert eigentlich bei der rekursiven Version?

Im Hauptprogramm steht der Aufruf:

```
System.out.println("\nDie Summe der Zahlen von 0 bis "+n+" lautet: "+summe(n));
```

Das Programm arbeitet nun bei der Eingabe von z. B. **n=3** wie folgt:



Beachte, dass wir in der rekursiven Version keine zusätzlichen Variablen benötigen, da der Compiler durch die wiederholten Methodenaufrufe stets dafür sorgt, dass bei jedem Methodenaufruf die aktuellen Werte aller Variablen auf dem internen Stack gespeichert werden. Es wird also schon Speicherplatz belegt, dieses passiert hier jedoch implizit, d.h. wir bekommen davon nichts mit.

Es soll hier nicht unerwähnt bleiben, dass bei rekursiven Programmen häufig (wie in dem Beispiel) sogar mehr Speicherplatz belegt wird als bei der entsprechenden iterativen Version. Auch das Laufzeitverhalten rekursiver Programm ist in der Regel schlechter, d.h., sie laufen aufgrund der vielen Methodenaufrufe länger, bis die Lösung gefunden ist.

Trotzdem sollte man, wenn das Problem es verlangt und der Algorithmus dadurch übersichtlicher wird, zu einer rekursiven Lösung greifen. Es gibt auch Probleme, wo die rekursive Lösung vermieden werden sollte, da sie einfach zu ineffizient ist. Dazu später mehr.

Das häufigste Problem beim Umgang mit Rekursion ist jedoch die Endlosschleife! Ein Problem wird ja nicht dadurch gelöst, dass man die Methode erneut aufruft. Man muss auch dafür sorgen, dass das Programm irgendwann stoppt.

Die Grundidee der Rekursion ist es, von einem Problem einen kleinen Teil sofort zu erledigen und das Restproblem, so es denn eine ähnliche Gestalt hat, durch einen Selbstaufruf zu lösen. Dabei ist darauf zu achten, dass in der Methode der Fall berücksichtigt wird, dass das Problem so klein ist, dass nichts mehr zu tun ist, damit das Programm stoppt (terminiert).

Aufgabe 1:

Schreibe eine iterative und eine rekursive Methode zur Berechnung der Fakultät.

Mathematische Definition iterativ

$$n! := n \cdot (n-1) \cdot (n-2) \dots 2 \cdot 1$$

Mathematische Definition rekursiv

$$n! := \begin{cases} n \cdot (n-1)! & \text{, falls } n > 1 \\ 1 & \text{, falls } n = 1 \end{cases}$$

Lade dir dazu die entsprechende Vorlage vom Server und fülle die markierten Stellen.