Objektorientiertes Programmieren (OOP)

07-Rekursion

Dr. Marcel Tilly

Bachelor Wirtschaftsinformatik, Fakultät Informatik

Rekursion

.skip[]

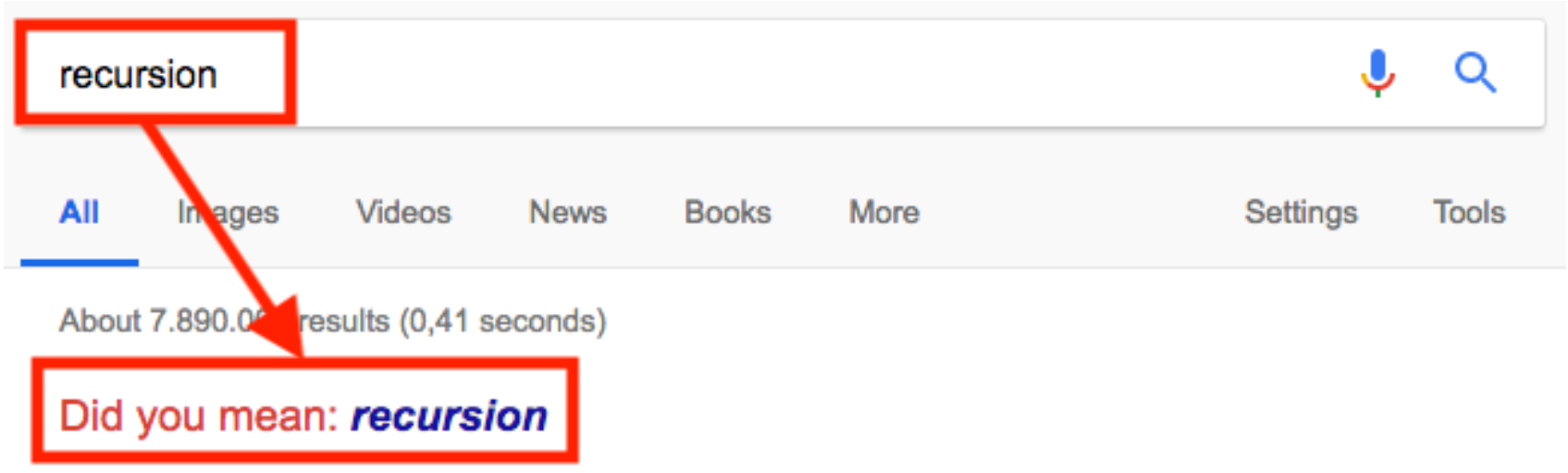


Figure 1: Rekursion

Fakultät

$$n! = \begin{cases} 1 & \text{für n} = 1 \text{ (terminal).} \\ n \cdot (n-1)! & \text{für n} > 1 \text{ (rekursiv).} \end{cases}$$

Fakultät

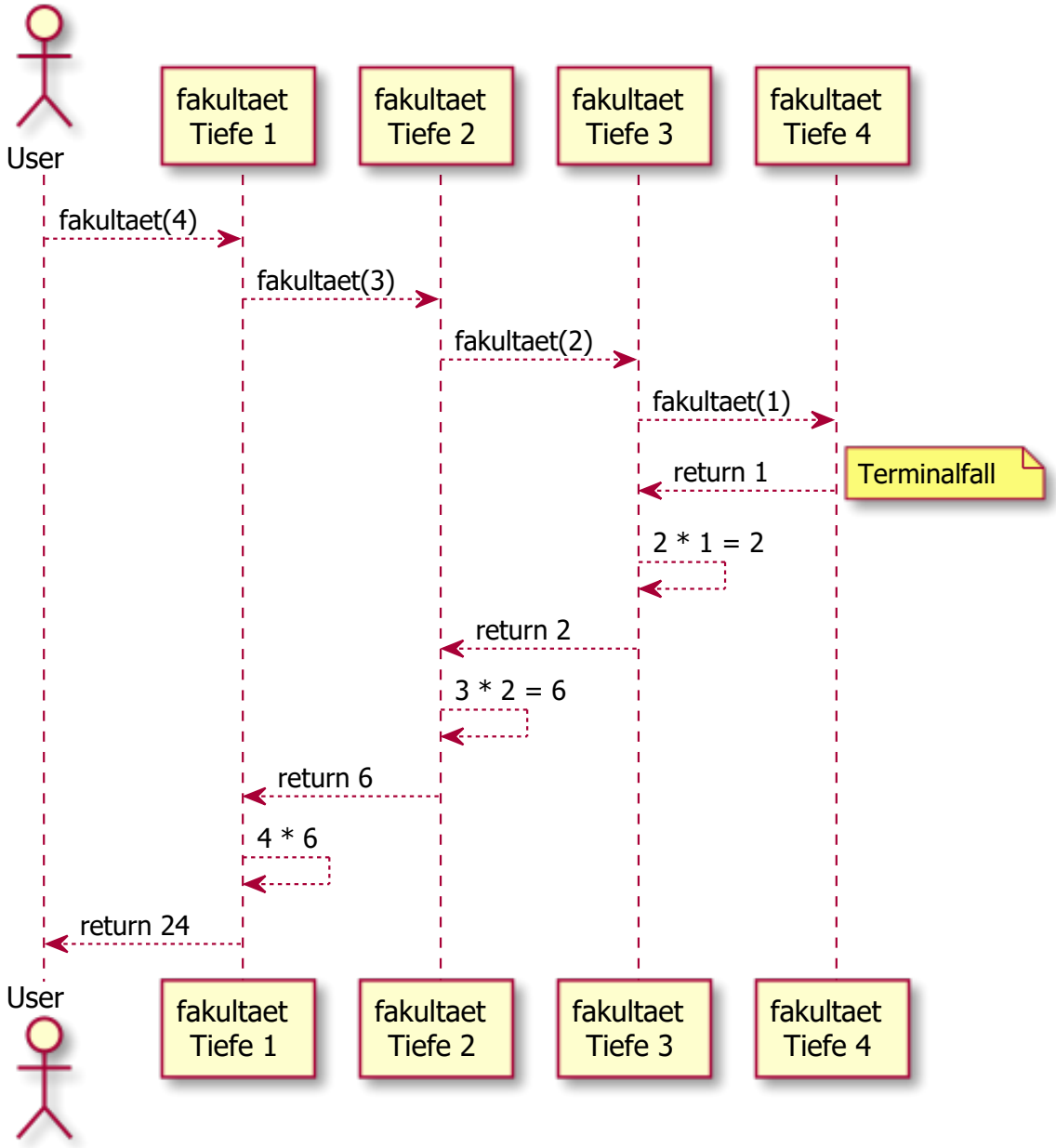


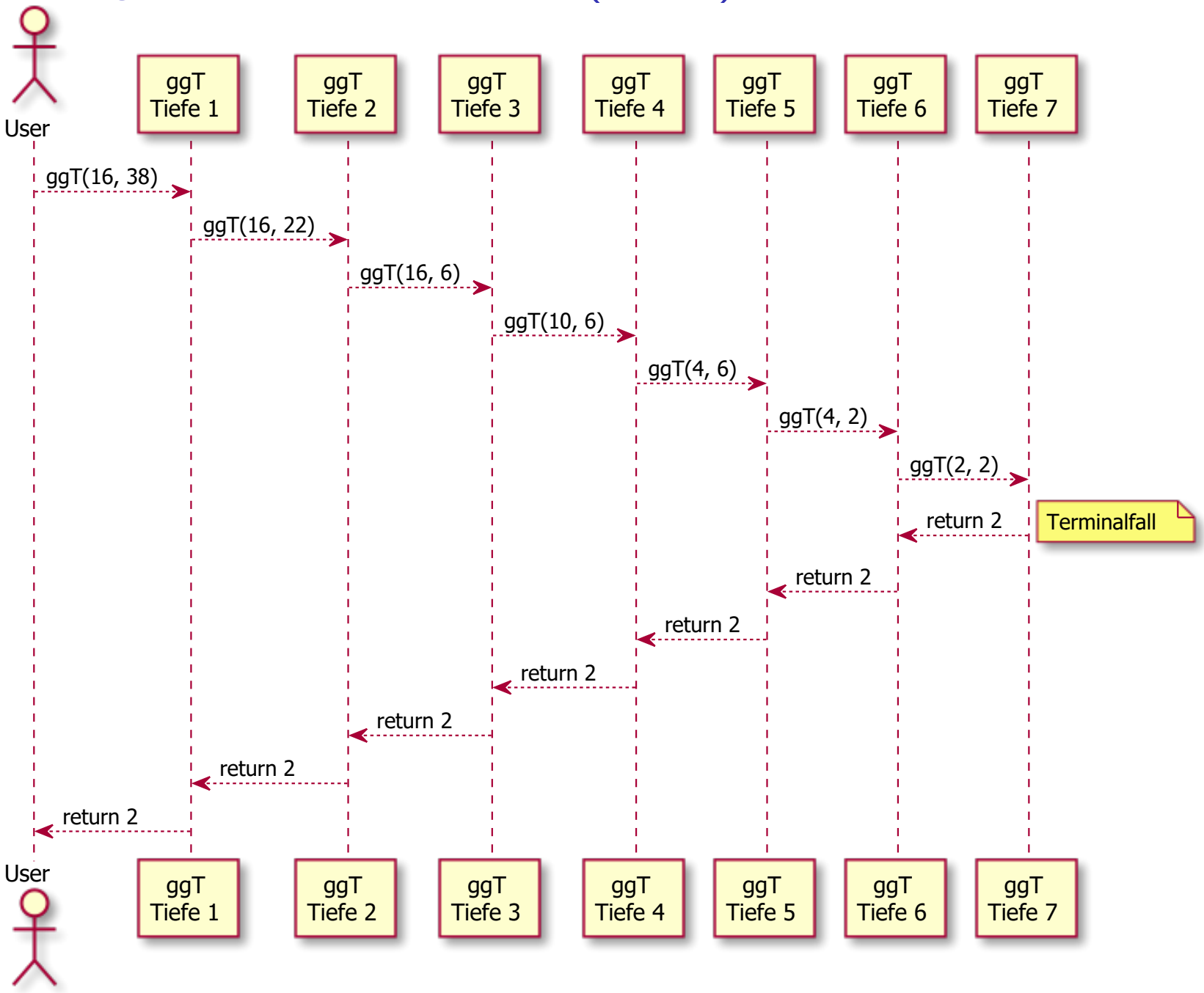
Figure 2: Fakultät, center

Größter gemeinsamer Teiler (GGT)

```
int ggT(int a, int b) {
    while (b != 0) {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }

    return a;
}
```

Größter gemeinsamer Teiler (GGT)



Fibonacci

$$\text{fib}(n) = \begin{cases} 0 & \text{für } n = 0 \\ 1 & \text{für } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{für } n > 1 \end{cases}$$

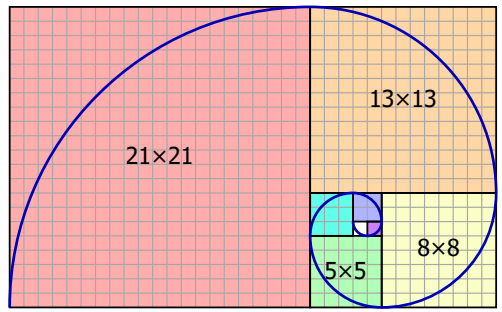
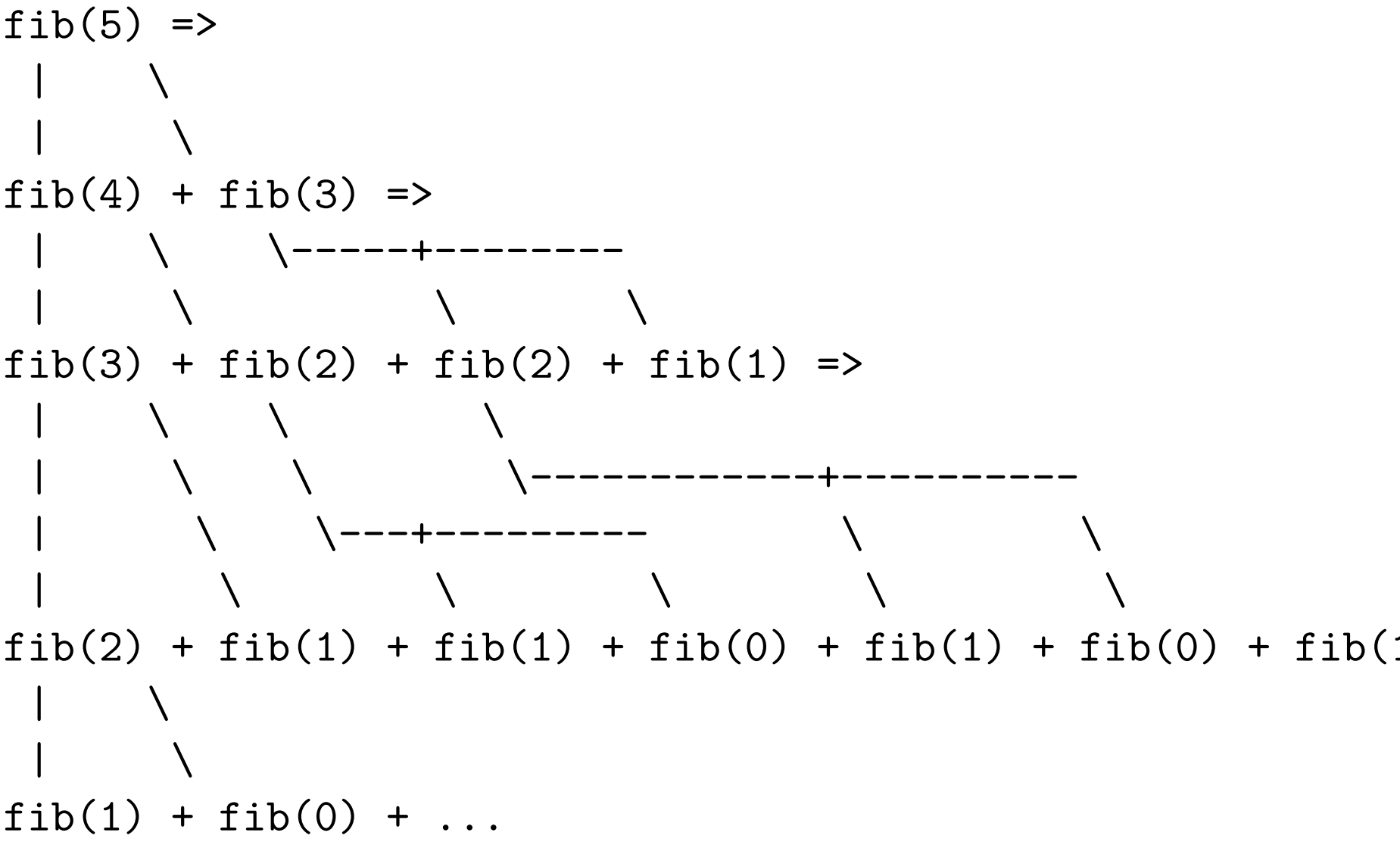


Figure 3: Fibonacci-Spirale, center

[Quelle: Wikipedia]

Fibonacci


fib(5) =>
fib(4) + fib(3) =>
fib(3) + fib(2) + fib(2) + fib(1) =>
fib(2) + fib(1) + fib(1) + fib(0) + fib(1) + fib(0) + fib(
fib(1) + fib(0) + ...

Kochrezept

1. Terminalfälle bestimmen. Wann ist die Lösung trivial?
2. Rekursionsfälle bestimmen. Wie kann ich das Problem auf ein kleineres runterbrechen?
3. Rekursion zusammensetzen: Brauche ich eine Hilfsmethode, wie muss die Signatur aussehen, wie müssen die Argumente beim rekursiven Aufruf verändert werden?

```
// kein valides Java...
int rekursiv(...) {
    if (Terminalfall) {
        return /* fester Wert */
    } else {
        // Rekursionsfall: mind. 1x rekursiv aufrufen!
        return rekursiv(/* veränderte Argumente*/);
    }
}
```

Arten der Rekursion

- **Lineare Rekursion:** genau ein rekursiver Aufruf, z.B. Fakultät.
- **Repetetive Rekursion** (Rumpfrekursion, engl. *tail recursion*): Spezialfall der linearen Rekursion, bei der der rekursive Aufruf die letzte Rechenanweisung ist. Diese Rumpfrekursionen können direkt in eine iterative Schleife umgewandelt werden (und umgekehrt). Beispiel: verbesserte Implementierung der Fibonacci Funktion.
- **Kaskadenartige Rekursion:** in einem Zweig der Fallunterscheidung treten *mehrere* rekursive Aufrufe auf, was ein lawinenartiges Anwachsen der Funktionsaufrufe mit sich bringt. Beispiel: einfache Implementierung der Fibonacci Funktion.
- **Verschränkte Rekursion:** Eine Methode `f()` ruft eine Methode `g()`, die wiederum `f()` aufruft.