

- **Abgabe:** Quellcode zu 6-1 und 6-2 auf Papier und ILIAS. Vorgegebene Dateien **nicht** ausdrucken!

Aufgabe 6-1

1. Schreiben Sie eine **rekursive** Methode `static long fib(int i)`, die die i -te Zahl der Folge

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, ...

berechnet. Z.B. soll der Aufruf `fib(7)` den Wert 13 liefern. Die erste Zahl der Folge ist 0 (entsprechend `fib(0)`), die zweite 1 (`fib(1)`), danach ist jede Zahl die Summe ihrer beiden Vorgänger, z.B. `fib(8) = fib(7) + fib(6)`. Allgemein: `fib(n) = fib(n-1) + fib(n-2)` (für $n \geq 2$)

Schreiben Sie dazu eine passende `main`-Methode, die die ersten 50 Zahlen der Folge am Bildschirm ausgibt. Was stellen Sie beim Ausführen des Programms fest?

2. Schreiben Sie eine **rekursive** Methode

```
public static long factorial(int n)
```

, welche die Fakultätsfunktion $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$ berechnet. Schreiben Sie dazu eine `main`-Methode, so dass man das Programm wie folgt von der Konsole aus starten kann:

```
$ java Factorial 10
3628800
```

Woran liegt es, dass dieses Programm effizienter ist als dasjenige aus Teilaufgabe 1? Überlegen Sie sich insbesondere, wieviele rekursive Methodenaufrufe für eine Berechnung nötig sind.

Hinweis: Die Funktion $n!$ wächst sehr schnell! Der Wert $21!$ übersteigt den Wertebereich von `long` bereits. Java ermöglicht das Rechnen mit sehr grossen natürlichen Zahlen durch die Klasse `BigInteger`.

3. Schreiben Sie ein **nicht-rekursives** Programm mit dem gleichen Output wie das Programm aus Teilaufgabe 1.
4. Gegeben sei folgende Methode:

```
static void iterative(){
    for(int i=0; i<=30; i+=3) System.out.println(i);
}
```

Schreiben Sie dazu eine äquivalente **rekursive** Methode (ohne Verwendung von Schleifen).

Überlegen Sie sich, wie *beliebige* Schleifen durch Rekursion eliminiert werden können.

Aufgabe 6-2

Implementieren Sie den rekursiven Mergesort-Algorithmus aus der Vorlesung. Schreiben Sie dazu eine Klasse `MergeSort` mit einer Methode

```
public static void sort(Comparable[] array)
```

, die einen Array von `Comparable`-Objekten sortiert. Schreiben Sie dazu eine rekursive Hilfsmethode `mergeSort(...)` und eine (nicht-rekursive) Hilfsmethode `merge(...)`.

Versuchen Sie die vorgegebenen Klassen `SortTest` und `Rectangle` zu verstehen und testen Sie Ihren Algorithmus mit der Klasse `SortTest`.

Hinweis: Ignorieren Sie allfällige Compilerwarnungen wie "`MergeSort.java uses unchecked or unsafe operations`".