Programmierkurs
Winter 2025/26

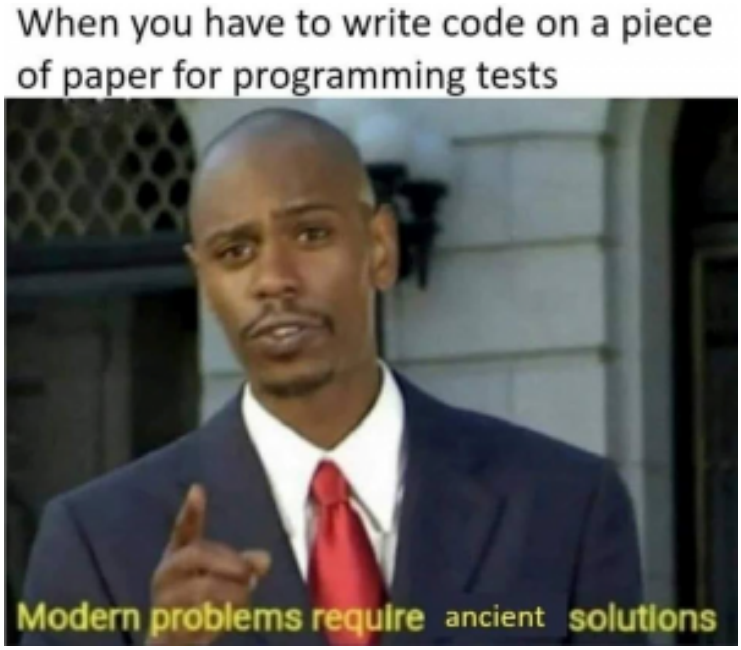
Department Mathematik / Informatik
Dr. Stefan Zellmann
Simon Wolf

Übungen 12.01-16.01.2026
Blatt 11

Abgabe der Aufgaben bis spätestens **Freitag, 16.01.2026 um 23:59 Uhr** via git.
Besprechung der Aufgaben in der Woche vom 19.01.2026.

Hinweise:

- Es gelten diese Modifikatoren-Regeln (es sei denn, es steht in der Aufgabe explizit anders):
  - Klassen: `public`
  - Objektattribute: `private`
  - Objektmethoden: `public` (insbesondere Getter, Setter und Konstruktoren)
- Konstruktoren: Implementieren Sie ausschließlich die von uns explizit verlangten Konstruktoren, und keine anderen.



Präsenzaufgabe 1 [**Potenzen rekursiv berechnen**]

Entwerfen Sie ein Programm, das Gleitkommazahlen ganzzahlig potenziert, das also a^b berechnet, wobei a eine Gleitkommazahl und b eine Ganzzahl ist.

(a) Das Programm soll rekursiv arbeiten.

(b) Das Programm soll iterativ arbeiten.

Präsenzaufgabe 2 [**Zweierlogarithmus**]

Berechnen Sie den Logarithmus zur Basis 2 einer Zweierpotenz. Gemeint ist damit also, dass bei der Zahl 16 die Antwort 4 ist, weil $2^4 = 16$. Bei 32 wäre die Antwort entsprechend 5, bei 1 wäre die Antwort 0. Entwerfen Sie ein Programm, das den Logarithmus zur Basis 2 einer Zweierpotenz ermittelt.

(a) Das Programm soll rekursiv arbeiten.

(b) Das Programm soll iterativ arbeiten.

Präsenzaufgabe 3 [**Kapitalanlage**]

Bei der Bank DreamyBank wird eine Kapitalanlage mit 5 Prozent verzinst. Das heißt, jedes Jahr wird das Kapital mitsamt Zinsen erneut für 5 Prozent verzinst, und die Zinsen dem Kapital gutgeschrieben. Weil derzeit die 5 Prozent ein sehr gutes Angebot sind, wird bei den resultierenden 5 Prozent Zinsen stets auf ganze Euro abgerundet. Bei einer Anlage von 1004 Euro hat sich also nach einem Jahr der Betrag 1054 Euro angesammelt (und nicht etwa 1054,20 Euro). Schreiben Sie ein rekursives Programm, das das angesammelte Kapital nach n Jahren errechnet.

Hausaufgabe 1 [Rekursiver Abstand] 20 Punkte

Erstellen Sie im Repository ein Package **`h1`** und fügen Sie diesem Package die Klasse **`H1_main`** hinzu, welche die **`main`**-Methode enthält.
Schreiben Sie die Klasse **`Node`**.

- Sie besitzt das **`Node`**-Attribut **`next`**, welches eine Referenz auf seinen Nachfolger speichert, sowie die dazugehörigen Getter und Setter.
- Sie besitzt den Konstruktor **`Node(Node next)`**, welche das oben genannte Attribut setzt.

Erzeugen Sie in der **`main`**-Methode eine Sequenz von Node (z.B. namens **`a,b,c,d,e`**), so dass in **`a`** das Attribut **`next`** eine Referenz auf **`b`** speichert (**`b`** ist also der Nachfolger von **`a`**), **`c`** der Nachfolger von **`b`**, und so weiter. Der Nachfolger von **`e`** ist **`null`**.

Ein Node hat zu sich selbst den Abstand 0. Ein Node hat zu seinem Nachfolger den Abstand 1. Ein Node hat zu dem Nachfolger seines Nachfolgers Abstand 2, und so weiter. In der oben genannten Sequenz hat zum Beispiel **`a`** den Abstand 3 zu **`d`**.

Schreiben Sie die Methode **`distance`** in die **`H1_main`**-Klasse.

- Modifikatoren: **`public static`**
- Rückgabetyp: **`int`**
- Name: **`distance`**
- Übergabeparameter: **`Node x, Node y`**

Diese Methode berechnet auf eine rekursive Weise den Abstand von **`x`** zu **`y`**. Sie dürfen davon ausgehen, dass **`x`** entweder gleich **`y`** ist oder in der Sequenz vor **`y`** vorkommt.

Hinweis:
Überlegen Sie sich den Base Case und den allgemeinen Fall. Notieren Sie sich Beispiele.

Hausaufgabe 2 [Fibonacci Folge] 40 Punkte

Die Fibonacci Folge ist rekursiv definiert als

$$F_n = F_{n-1} + F_{n-2}, \text{ wobei } F_1 \text{ und } F_2 = 1 \text{ und } n \in \mathbb{N}$$

Erstellen Sie im Repository ein Package **`h2`** und fügen Sie diesem Package die Klasse **`H2_main`** hinzu, welche die **`main`**-Methode enthält.

Im folgenden werden drei Methoden geschrieben, um die Folge zu berechnen. Messen Sie im folgenden die Ausführungszeiten aller drei Methoden und schreiben Sie dafür eine weitere öffentliche Klassenmethode **`benchmark(int n)`**, die nacheinander die drei Fibonacci Methoden für **`n`** aufruft und die vergangene Zeit ausgibt. Dazu merken Sie sich in **`main`** mit Hilfe der Bibliotheksmethode **`System.nanoTime()`** die Zeit, die vor und nach dem jeweiligen Methodenaufruf

vergangen ist. Die Differenz daraus gibt Aufschluss darüber, wie viel Zeit die Methode gebraucht hat. Geben Sie diese jeweils auf der Konsole aus, z. .B.: `Elapsed nanoseconds (methodName): 12345678`. **`benchmark`** können Sie nun aus Ihrer **`main`** Methode heraus aufrufen.

(a) Schreiben Sie die Klassenmethode **`fibonacci`**, die die Fibonacci Folge rekursiv für Ganzzahlen **`n`** berechnet und das Ergebnis zurückgibt. Geben Sie das Ergebnis sowie die Ausführungszeit (s. o.) auf der Konsole aus.

(b) Die rekursive Methode aus (a), wenn Sie sie korrekt implementiert haben, berechnet die gleichen Zwischenergebnisse mehrfach. Schreiben Sie nun auf Basis von (a) die rekursive Methode **`fibonacciCached`**. Diese Methode verwendet ein globales Array konstanter Länge (z. B. 1000 Einträge), um sich Fibonacci Zahlen, die während der Ausführung bereits berechnet wurden, zu merken. Findet sich ein Eintrag bereits in diesem Array, wird dieser zurückgegeben, anstatt das Zwischenergebnis rekursiv zu berechnen. Geben Sie auch hier das Ergebnis sowie die Ausführungszeit auf der Konsole aus.

(c) Schreiben Sie nun eine Methode, die die Fibonacci-Folge für Ganzzahlen **`n`** iterativ, z. B. mit Hilfe einer **`for`**-Schleife ausrechnet. Wieder geben Sie das Ergebnis und die Ausführungszeit auf der Konsole aus.

Hausaufgabe 3 [**Merge Sort**] **40 Punkte**

Erstellen Sie im Repository ein Package **`h3`** und fügen Sie diesem Package die Klasse **`H3_main`** hinzu, welche die **`main`**-Methode enthält.
Fügen Sie der Klasse eine Klassenmethode **`mergeSort`** hinzu, die eine Referenz auf ein Array mit Elementen vom Typ **`int`** übernimmt und eine Referenz auf ein sortiertes Array vom selben Typ zurückgibt. Implementieren Sie Merge Sort rekursiv wie in der Vorlesung. Achten Sie darauf, dass **`mergeSort`** Arrays beliebiger Länge sortieren können sollte und nicht nur solche, deren Länge einer ganzzahligen Zweierpotenz entspricht.