

DAP2 Praktikum – Blatt 1

Abgabe: ab 11. April

Studienleistung

- Zum Bestehen des Praktikums muss jeder Teilnehmer die folgenden Leistungen erbringen:
 - Es müssen mindestens 50 Prozent der Punkte in den Kurzaufgaben erreicht werden.
 - Es müssen mindestens 50 Prozent der Punkte in den Langaufgaben erreicht werden.
- Im Krankheitsfall kann ein Testat bei Vorlage eines Attests in der folgenden Woche nachgeholt werden.
- Wenn ein Praktikumstermin auf einen Feiertag fällt, müssen Sie sich an einem beliebigen anderen Praktikumstermin in der gleichen Woche testieren lassen.
- **Hinweis:** Notieren Sie sich Ihre Punkte nach jedem Testat! Dies dient der eigenen Kontrolle. (Ihr Punktestand kann Ihnen während des Semesters nicht genannt werden kann.)

Wichtige Information (im Moodle verfügbar)

- Beachten Sie die Erklärung des **Ablaufs (Blatt A)**.
- Beachten Sie die **Regeln und Hinweise (Blatt R)** in der aktuellsten Version!
- Beachten Sie die **Hilfestellungen (Blatt H)** in der aktuellsten Version!

Kurzaufgabe 1.1: Kommandozeile & ArrayLists

(4 Punkte)

In dieser Aufgabe üben Sie, wie man Eingabedaten mit Java verarbeiten kann. Bitte beachten Sie dazu insbesondere die Regeln auf Blatt R, und die Hilfestellungen auf Blatt H.

- (a) Kompilieren Sie die Beispiel-Programme zu Command-Line-Argumenten und Standard-In von **Blatt H**. Probieren Sie die verschiedenen Eingabemöglichkeiten für Standard-In aus, bis Sie das Gefühl haben, den Mechanismus verstanden zu haben **(0 Punkte)**.
- (b) Schreiben Sie ein Programm, das eine Liste von Ganzzahlen via Standard-In erhält, und diese in einem Array vom Typ `int []` ablegt. Da Sie beim Lesen von Standard-In nicht wissen, wie viele Elemente noch folgen, sollten Sie die Zahlen zunächst in einer `ArrayList<Integer>` zwischenspeichern (die Verwendung von `ArrayList` ist hier explizit erlaubt). Der `ArrayList` können dynamisch Elemente hinzugefügt werden. Sobald das Lesen der Eingabe abgeschlossen ist, können Sie die `ArrayList` händisch in ein Array konvertieren. **(2 Punkte)**
- (c) Schreiben Sie ein Programm, das eine Liste von Ganzzahlen a_0, a_1, \dots, a_{n-1} via Standard-In sowie eine positive Ganzzahl k als Argument erhält, und den k -kleinsten Wert in der Liste als Ergebnis ausgibt. Sie dürfen dazu ihre Lösung von Teilaufgabe (b) und die Methode `Arrays.sort(...)` aus der Bibliothek `java.util.Arrays` verwenden, welche ein Array aufsteigend sortiert. **(2 Punkte)**. Beispiel:

```
seq 13 37 | shuf | java Select 3
```

```
The 3-smallest value is 15.
```

```
seq 11 18 | java Select 10
```

```
The list must contain at least k=10 numbers, but only 8 were provided.
```

Kurzaufgabe 1.2: Enumerierung von Permutationen (4 Punkte)

Schreiben Sie ein Programm, das eine Liste von Ganzzahlen a_0, a_1, \dots, a_{n-1} via Standard-In als Eingabe erhält, und alle Permutationen von a_0, a_1, \dots, a_{n-1} ausgibt (eine Permutation pro Ausgabezeile). Am Ende soll die Anzahl der Permutationen ausgegeben werden. Beispiel:

```
seq 11 3 17 | java EnumeratePermutations
[11, 14, 17]
[11, 17, 14]
[14, 11, 17]
[14, 17, 11]
[17, 11, 14]
[17, 14, 11]
```

Es gibt 6 Permutationen der Eingabe.

Sie dürfen davon ausgehen, dass alle Eingabewerte unterschiedlich sind (ohne dies im Programm zu überprüfen). Es spielt keine Rolle, in welcher Reihenfolge Sie die Permutationen ausgeben. Bei der Implementierung sollten Sie das Konzept der *Rekursion* verwenden, welches Sie bereits aus DAP1 kennen. Eine geeignete Signatur für die rekursive Methode sieht wie folgt aus:

```
public static int printPermutations(int [] array, int d)
```

In der `main`-Methode lesen Sie einfach die Eingabe wie in Aufgabe 1.1(b) ein, und rufen dann `printPermutations` mit `d = 0` auf.

Sie dürfen keine Hilfsarrays verwenden (die Eingabewerte müssen *in-place* permutiert werden). Zur Ausgabe der Permutationen dürfen Sie die Methode `Arrays.toString()` aus der Bibliothek `java.util.Arrays` verwenden.

Wie Ihnen vermutlich bekannt ist, gibt es $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$ verschiedene Permutationen einer n -elementigen Menge. So gibt es z.B. $5! = 120$ Permutationen einer 5-elementigen Menge. Ihr Programm sollte keine Permutation doppelt ausgeben, und zusätzlich zur Ausgabe nur $\mathcal{O}(n!)$ Rechenschritte benötigen.

Tip: Das Argument d gibt an, wie viele Positionen im Array bereits *festgelegt* sind. Beim Aufruf von `printPermutations` haben Sie sich bereits für die ersten d Elemente entschieden. Jetzt müssen Sie lediglich rekursiv alle Permutationen auflisten, die mit diesen d Elementen beginnen. Beispiele zum Aufruf von `printPermutations`:

```
printPermutations([5,11,14,17], 3) (Rückgabewert: 1)
[5, 11, 14, 17]
```

```
printPermutations([5,11,14,17], 2) (Rückgabewert: 2)
[5, 11, 14, 17]
[5, 11, 17, 14]
```

```
printPermutations([5,11,14,17], 1) (Rückgabewert: 6)
[5, 11, 14, 17] } = Ausgabe von printPermutations([5,11,14,17], 2)
[5, 11, 17, 14] }
[5, 14, 11, 17] } = Ausgabe von printPermutations([5,14,11,17], 2)
[5, 14, 17, 11] }
[5, 17, 14, 11] } = Ausgabe von printPermutations([5,17,14,11], 2)
[5, 17, 11, 14] }
```