Übungsblatt 2 – Lösungen

Die Java-Programme finden sich in den zugehörigen `.java`-Dateien.

Aufgabe 1 - Größter echter Teiler

Konzipiere einen effizienten Algorithmus, der für einen gegebenen Wert `n>1` den größten echten Teiler `z` bestimmt.
Für einen echten Teiler `z` gilt: `n%z==0` und `n!=z`.
Implementiere die Methode `int greatestProperDivisor( int n )`, die Deinen Algorithmus umsetzt.
Hinweis: Für Primzahlen ist der größte echte Teiler immer `1`.

Lösung:

- Der größte echte Teiler ist auch für negative Werte immer positiv.
- Effizient ist die Suche nach dem *kleinsten* echten Teiler: Der größte echte Teiler ist dann der Co-Teiler.
  Da die kleinen Werte viele Werte teilen, wird der kleinste echte Teiler schnell gefunden.
- Die Suche nach dem kleinsten echten Teiler kann bei $\lfloor n/2 \rfloor$ abgebrochen werden.

Aufgabe 2 - Suche von Paaren in Feldern

Implementiere eine Methode `int countPairs( int[] arr )`, die die Anzahl der Paare von zwei unmittelbar aufeinander folgenden gleichen Werten in einem Feld bestimmt. Dabei soll jedes Element des Feldes *nur zu genau einem Paar* gehören.
Beispiele:

`1 1 3 3 1 2 2 2 1` – besitzt drei Paare: `11, 33, 22`
`1 3 3 3 3 2 2 5 5` – besitzt vier Paare: `33, 33, 22, 55`

Aufgabe 3 - Klausurenstapel

Auf dem Übungsblatt 1 hast Du in Aufgabe 3 verschiedene Algorithmen konzipiert, die nun in Java implementiert werden sollen. Gehe in dieser Aufgabe davon aus, dass für jeden der beiden Klausurstapel ein (unsortiertes) Feld mit den zugehörigen Matrikelnummern zur Verfügung steht.
Implementiere die folgende Methoden:

- `boolean writtenBoth( int[] exam1, int[] exam2 )` gibt `true` zurück, falls es mindestens eine Matrikelnummer gibt, die sowohl in `exam1` als auch in `exam2` vorkommt. sonst wird `false` zurückgegeben.
- `boolean notWrittenBoth( int[] exam1, int[] exam2 )` gibt `true` zurückgeben, wenn es keinen Studierenden gibt, der beide Klausuren mitgeschrieben hat. Ansonsten wird `false` zurückgegeben.
- `int countWrittenBoth( int[] exam1, int[] exam2 )` ermittelt die Anzahl an gleichen Matrikelnummern in `exam1` und `exam2`.
- `boolean biggerThan( int[] exam1, int[] exam2 )` gibt `true` zurück, wenn alle Matrikelnummer in exam1 größer sind als in `exam2`, sonst `false`.
- *Zusatzaufgabe:* Implementiere Varianten für die oben beschriebenen Methoden unter der Annahme, dass die Felder `exam1` und `exam2` aufsteigend sortiert sind.

Aufgabe 4 - Primfaktorzerlegung

In der Vorlesung sind Methoden vorgestellt worden, mit denen eine Folge der Primzahlen bis zu einer Obergrenze `n` erzeugt werden kann. Implementiere die Methode `void primeFactorization( int x )`, die auf Basis dieser Primzahlen für eine vorgegebene Zahl `x>0` deren Zerlegung in Primfaktoren bestimmt und ausgibt.
Beispiele:

```
204 = 2 * 2 * 3 * 17
150 = 2 * 3 * 5 * 5
273 = 3 * 7 * 13
```

Lösung:
Die Primzahlen können mit den aus der Vorlesung bekannten Methoden initializeNumbers, inspectNumbers und primeNumbersToArray bestimmt und in einem Feld abgelegt werden. Die so erzeugten Primzahlen werden nacheinander als Teiler getestet. Ist eine Primzahl Teiler, wird sie erneut getestet, sonst wird mit der nächsten Primzahl fortgefahren.

Aufgabe 5 - Zweidimensionale Felder (Matrix-Bearbeitung)

In der Vorlesung wurden Felder mit Grundtypen `int` oder `boolean` eingeführt. Es ist aber auch möglich, als Grundtyp eines Feldes wiederum einen Feld-Typ zu wählen: Es entstehen so Felder, deren Elemente selbst wiederum Felder sind.
Da sich eine solche Struktur gut zweidimensional visualisieren lässt, spricht man auch von *zweidimensionalen Feldern*.

- Implementiere die Methode `boolean isSquare( int[][] matrix )`, die als Parameter ein zweidimensionales Feld mit `int`-Werten übergeben bekommt und prüft, ob dieses die gleiche Anzahl von Zeilen und Spalten besitzt.
- Implementiere die Methode `void scalMult( int[][] matrix, int k )`, die als Parameter ein zweidimensionales, quadratisches Feld `matrix` mit `int`-Werten und einen einzelnen `int`-Wert `k` übergeben bekommt und jeden `int`-Wert des übergebenen Feldes mit `k` multipliziert.
- Implementiere die Methode `int[] extractDiagonal( int[][] matrix )`, die als Parameter ein zweidimensionales, quadratisches Feld `matrix` mit `int`-Werten übergeben bekommt und ein Feld der Länge `m` zurückgibt, das alle Werte der Diagonalen `matrix[1][1], matrix[2][2], ... matrix[m][m]` enthält.

Lösung:
Da in Java zweidimensionale Felder technisch als *Felder-von-Feldern* realisiert werden, enthält das strukturbildende (äußere) Feld Referenzen auf eigenständige Felder, die insbesondere nicht die gleiche Länge besitzen müssen. Soll sichergestellt werden, dass ein Feld qudratisch oder zumindest rechteckig ist, so muss dieses in der Konstruktion oder durch Überprüfungen – wie mit der Methode `isSquare` – explizit abgesichert werden.