Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Institut für Angewandte und Numerische Mathematik

Programmieren: Einstieg in die Informatik mit Java

WS 2006/2007

Dr. G. Bohlender
Dipl.–Math. techn. M. Richter

06.11.2006

Aufgabenblatt 2

Bearbeitungszeitraum: 09.11.2006 – 22.11.2006

Aufgabe 4 (Pflichtaufgabe): *Summation von Zahlen*

(a) Erstellen Sie ein Java-Programm mit dem Namen `Summation`, welches eine positive, ganze Zahl n (d.h. einen Wert vom Typ `int`) einliest und anschließend die Summe

$$S(n) = \sum_{k=1}^{n} k = 1 + 2 + 3 + 4 + \cdots + n$$

der ersten n natürlichen Zahlen **mit einer `for`-Schleife** berechnet. Geben Sie das Ergebnis auf dem Bildschirm aus.

(b) Berechnen Sie außerdem **mit einer `while`-Schleife** die Summe

$$U(n) = \sum_{k=1}^{n} (2k-1) = 1 + 3 + 5 + \cdots + (2n-1)$$

der ersten n ungeraden Zahlen und geben Sie das Ergebnis am Bildschirm aus.

Hinweis: Natürlich können die Summen $S(n)$ und $U(n)$ auch direkt berechnet werden. Es gilt nämlich z.B. die Gaußsche Summationsregel $S(n) = \frac{1}{2}n(n+1)$. Durch die Berechnung von $U(1), U(2), U(3), U(4), \ldots$ kann man außerdem leicht eine Summationsregel für $U(n)$ finden und diese anschließend mittels vollständiger Induktion beweisen (keine Pflicht!). Um ein Testat für diese Aufgabe zu erhalten, müssen Sie jedoch $S(n)$ und $U(n)$ mit einer `for`- bzw. einer `while`-Schleife berechnen.

Aufgabe 5: *Pythagoräische Tripel*

Schreiben Sie ein Java-Programm mit dem Namen `PythagoraeischeTripel`, welches durch reines Probieren alle *pythagoräischen Tripel* bis zu einer oberen Grenze N findet. Gesucht sind also alle Tripel (a, b, c) natürlicher Zahlen (d.h. vom Typ `int`), bei denen die Summe der Quadrate der ersten beiden Zahlen a und b das Quadrat der dritten Zahl c ergibt:

$$a^2 + b^2 = c^2.$$

Lesen Sie dazu die obere Grenze N ein. Verwenden Sie **drei ineinander geschachtelte `for`-Schleifen** und überprüfen Sie die Bedingung $a^2 + b^2 = c^2$ in einer `if`-Anweisung. Achten Sie bei der Definition der `for`-Schleifen darauf, daß alle Zahlenkombinationen (a, b, c) nur **einmal** überprüft werden. Geben Sie die gefundenen pythagoräischen Zahlentripel auf dem Bildschirm aus. Die ersten vier Tripel lauten wie folgt:

```
3 4 5
6 8 10
5 12 13
9 12 15
```

Aufgabe 6 (Pflichtaufgabe): *Primzahltest*

Im modernen elektronischen Zahlungsverkehr sorgen sogenannte *public-key*-Verschlüsselungsalgorithmen für eine sichere Datenübertragung zwischen Banken und ihren Kunden. Ein wichtiger Algorithmus, der hierbei zum Einsatz kommt, ist der RSA-Algorithmus (RFC 2437). Für diesen Algorithmus werden große Primzahlen benötigt (mit ca. 200 Dezimalstellen). Solche Primzahlen können i.d.R. nur mit leistungsfähigen Rechnern gefunden werden, die einen bestimmten Zahlenbereich nach Primzahlen absuchen.
In dieser Aufgabe soll ein sehr einfacher Primzahltest realisiert werden. Genauer gesagt soll ein Programm geschrieben werden, welches zu einer natürlichen Zahl N die kleinsten echten Teiler ungleich Eins sucht. Kann ein solcher Teiler nicht gefunden werden, so handelt es sich bei N um eine Primzahl.

(a) Erstellen Sie ein Java-Programm mit dem Namen `Primzahltest`, welches zunächst eine ganze Zahl der Größe 64 Bit einliest. Definieren Sie dazu eine Variable `zahl` vom Typ `long` und lesen Sie mit der Anweisung

```
zahl = sc.nextLong();
```

einen Wert von der Konsole ein.

(b) Definieren Sie drei Variablen `teiler`, `divisor` vom Typ `long`. Initialisieren Sie die Variable `teiler` mit dem Wert `1` und `divisor` mit dem Wert `2`. Speichern Sie in der Variable `maxdivisor` die **gerundete Quadratwurzel** von `zahl` ab. Verwenden Sie dazu die Methoden `Math.sqrt()` und `Math.round()`.

(c) Schreiben Sie eine `do-while`-Schleife, in der folgendes geschieht: Zunächst wird getestet, ob `zahl` durch `divisor` teilbar ist. Falls dies der Fall ist, wird der Wert von `divisor` in `teiler` abgespeichert. Anschließend wird `divisor` um Eins erhöht. Die Schleife soll solange durchlaufen werden, wie der Wert von `divisor` kleiner oder gleich `maxdivisor+2` ist, und solange der Wert von `teiler` gleich Eins ist.

Hinweis: Verwenden Sie eine `if`-Anweisung. Eine ganze Zahl `z` ist genau dann durch eine zweite ganze Zahl `t` teilbar, wenn die Operation `z % t` das Ergebnis Null hat.

(d) Besitzt `teiler` nach dem Schleifendurchlauf einen Wert, der größer als Eins ist, so ist `zahl` durch diesen Wert teilbar. Andernfalls handelt es sich bei `zahl` um eine Primzahl. Geben Sie je nach Ergebnis des Primzahltests eine Meldung auf dem Bildschirm aus. Verwenden Sie dabei eine `if-else`-Anweisung.

(e) Testen Sie Ihr Programm mit bekannten Primzahlen (z.B. `7`, `23`, `911`) sowie mit bekannten zusammengesetzten Zahlen (z.B. `15`, `39`, `121`). Entscheiden Sie anschließend, ob die folgenden Zahlen Primzahlen oder zusammengesetzte Zahlen sind:

```
1254252341727733
2436767952612529
3514769285939839
7351274923758263
```