# 1. Aufgabe

Leonardo Fibonacci hat 1202 die Frage nach der Anzahl an Kaninchenpärchen untersucht. Unter der Annahme, dass im Jahr 1 mit einem Paar begonnen wird, jedes Pärchen vom zweiten Jahr an ein weiteres Pärchen Nachwuchs bekommt und die Kaninchen unendlich lange leben, wieviele Paare gibt es nach n Jahren?

Die Populationsgröße kann in diesem Fall durch die sogenannten Fibonacci-Zahlen ausgedrückt werden und zwar rekursiv durch:

$$f(n) = \begin{cases} 0 & \text{für } n = 0 \\ 1 & \text{für } n = 1 \\ f(n-1) + f(n-2) & \text{für } n > 1 \end{cases}$$

(a) Schreiben Sie ein Java-Programm, welches in der main-Methode die Fibonacci-Zahlen auf zwei Arten löst, und zwar sowohl rekursiv als auch iterativ. Dazu soll eine beliebige Eingabe von der Tastatur empfangen und die beiden Ergebnisse auf den Bildschirm ausgegeben werden.

(b) Erweitern Sie das Programm unter Teilaufgabe (a) so, dass nun auch Performance-Vergleiche möglich sind. Berechnen Sie dazu die für beide Alternativ- Lösungen die Systemzeit und geben Sie beide Zeiten zum Vergleich auf dem Bildschirm aus. Verwenden Sie dazu das Objekt System aus dem (implizit bekannten) Package java.lang. Der entsprechende Programmcode sollte sich an dem folgenden Beispiel orientieren:

```
long zeitpunktStart = System.nanoTime ();
result = berechnung (... );
long zeitpunktEnde = System.nanoTime ();
long dauer = zeitpunktEnde - zeitpunktStart;
```

Für weitere Details konsultieren Sie die entsprechende API des Objekts System aus dem java.lang-Package.

(c) Auf der Basis des in Teilaufgabe (b) entwickelten Programmcodes sollen nun einige Testreihen durchgeführt werden. Starten Sie zunächst mit kleinen Eingabewerten und beobachten Sie die entsprechenden Rechenzeiten. Gibt es einen maximalen Wert, der nicht überschritten werden kann und wenn ja, erläutern Sie den Hintergrund!

**Prof. Dr. M. Duque-Antón**

# 2. Aufgabe

Als nächste Anwendung der Rekursion betrachten wir das Pascalsche Dreieck. Dieses sieht folgendermaßen aus und gibt die Koeffizienten in den binomischen Formeln an:

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5  10  10   5   1   usw.
```

Am Rand stehen jeweils „Einsen“ und in der Mitte ergibt sich jede Zahl aus der Summe der beiden benachbarten Zahlen der vorherigen Zeile. Wir bezeichnen die Zeilen mit n und die Reihenfolge der Zahlen innerhalb einer Zeile mit k. Die Zählung beginnt dabei sowohl bei den Zeilen als auch den „Spalten“ mit Null; k kann jeweils alle Werte zwischen 0 und dem aktuellen n annehmen. Wir bezeichnen die Zahlen auch als Binomialkoeffizienten B(n,k) und erhalten beispielsweise:

B(0,0) = 1, B(2,1) = 2; B(4,2) = 6 oder B(5,2) = B(4,1) + B(4,2) = 4 + 6 = 10

Allgemein lautet das Bildungsgesetz:

$$B(n,k) = \begin{cases} 1 & \text{für } k = 0 \text{ oder } k = n \\ B(n-1,k-1) + B(n-1,k) & \text{sonst} \end{cases}$$

Man kann in dieser Form den rekursiven Ansatz leicht erkennen: bei der Berechnung von B(n,k) wird auf die Werte für kleinere Argumente n und k zurückgegriffen. Bitte implementieren Sie ein Java-Programm, welches das Pascalsche Dreieck auf dem Bildschirm ausgibt und zwar in der üblichen Dreiecks-Form. Die Eingabe erfolgt von der Tastatur. Für die Anzahl n der Zeilen soll eine Beschränkung eingefügt werden, damit die Darstellbarkeit auf dem Bildschirm noch garantiert werden kann.

# 3. Aufgabe

Wir betrachten nun das Wechselgeld-Beispiel aus Teilkapitel 6.3. Es soll ein Greedy-Algorithmus in Java implementiert werden, der auf Geldbeträge unter 1 EUR Wechselgeld herausgibt. Zur Verfügung stehen ausreichend Münzen mit den Werten 50, 10, 5, 2 und 1 Cent. Das Wechselgeld soll aus so wenig Münzen wie möglich bestehen. Das Programm soll den Betrag (in Cent) von der Tastatur lesen und das Wechselgeld auf den Bildschirm ausgeben.

# 4. Aufgabe

Wir betrachten nun das Turnierplan-Beispiel aus Teilkapitel 6.3. Es soll ein Divide-and-Conquer-Algorithmus in Java implementiert werden, welcher die Anzahl der Spieler als Exponent von der Tastatur erwartet und den entsprechenden Turnierplan berechnet und auf dem Bildschirm ausgibt. Es werden also nur 2er Potenzen von Spieleranzahlen zugelassen. Die Anzahl der Spieler beträgt also $2^{exponent}$ und die Anzahl der Spieltage $2^{exponent} - 1$. Der auf den Bildschirm darzustellende Turnierplan soll als Matrix mit $2^{exponent}$ Zeilen und $2^{exponent} - 1$ Spalten ausgegeben werden. Verifizieren Sie die Ergebnisse zumindest bis zum Exponenten 4, also für einen Turnierplan mit 16 Spielern und 15 Spieltagen.

# 5. Aufgabe

Gegenstand dieser Aufgabe ist das N-Damen-Problem aus Teilkapitel 6.3. Es soll ein Backtracking-Algorithmus in Java implementiert werden, welcher die generische Schachbrett-Größe als Eingabe erwartet und alle korrekten Damen-Belegungen auf dem Bildschirm ausgibt. Das Ergebnis für die Schachbrett-Größe N = 4 soll manuell überprüft werden und mit dem vom Programm berechneten Ergebnis verglichen werden. Bitte wenden Sie Ihr Programm auch auf die Größe N = 8 an. Vergleichen Sie dazu stichprobenartig die mit Hilfe der Systematik bzw. Programm ermittelte erste und letzte Lösung (Damenbelegungen). Wieviele Lösungen gibt es für N = 8?