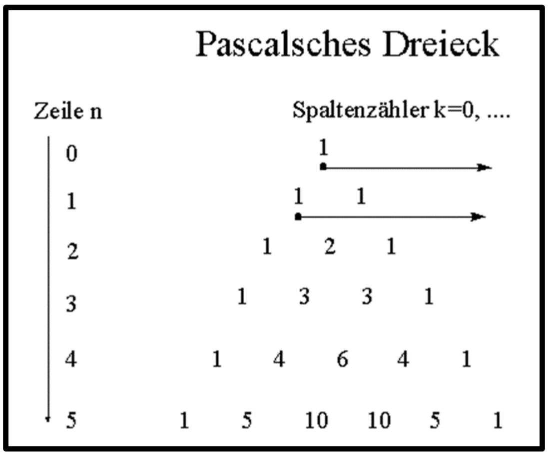
Aufgabe 1 (LK-Klausur 1819)

Bei der Berechnung von Termen der Art $(a+b)^n$ $(n \in \mathbb{N})$ treten Koeffizienten auf, die man mit dem sog. Pascalschen Dreieck berechnen kann (siehe Abbildung). Die einzelnen Koeffi-zienten des Pascalschen Dreiecks werden mit B(n;k) bezeichnet, wobei n die Zeile und k die Spalte angeben (beide Zählungen beginnen bei 0). Demnach gilt: B(3;1) = 3 oder B(4;4) = 1.



a) Geben Sie die Werte für B(2;0) und B(5;4) an.

Beschreiben Sie, wie eine Zeile n aus der vorherigen Zeile berechnet wird und geben Sie die Koeffizienten der Zeile für n = 7 an.

Zwei mögliche Definitionen für die Berechnung der Werte für B sind gegeben durch:

[1] $$B(n;k) = \begin{cases} 1 & k = 0 \lor k = n \\ B(n-1;k) + B(n-1;k-1) & sonst \end{cases}$$

[2] $B(n;k) = \frac{n!}{k! \cdot (n-k)!}$ (n! in Java: `Math.fac(n)`)

b) Beschreiben Sie die Arbeitsweise der beiden Definitionen und berechnen Sie mit beiden Definitionen (mit einer Gleichung oder einem Aufrufbaum) den Wert für B(4;2).

c) Implementieren Sie eine rekursive Funktion `bRek(int n, int k)`, welche die Koeffizienten für das Pascalsche Dreieck berechnet.

Beschreiben Sie Vor- und Nachteile einer rekursiven gegenüber einer iterativen Methode.

Für Terme der Art $(a+b)^n$ $(n \in \mathbb{N})$ gilt:

$$(a+b)^n = B(n;0) * a^n b^0 + B(n;1) * a^{n-1} b^1 + B(n;2) * a^{n-2} b^2 \ldots + B(n;n) * a^0 b^n$$

Damit gilt beispielsweise:

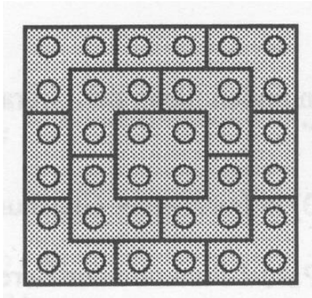
$$(a+b)^2 = B(2;0) * a^2 b^0 + B(2;1) * a^1 b^1 + B(2;2) * a^0 b^2 = a^2 + 2ab + b^2$$

$$(a+b)^3 = B(3;0) a^3 b^0 + B(3;1) a^2 b^1 + B(3;2) a^1 b^2 + B(3;3) a^0 b^3 = a^3 + 3a^2 b + 3ab^3 + b^3$$

d) Leiten Sie den Term für $(a+b)^5$ her.

Implementieren Sie eine Methode `term(int n)`, welche den Term für $(a+b)^n$ in einem String zurückgibt. Hinweise: Die Faktoren 1 und die Exponenten 0 bzw. 1 müssen nicht vereinfacht werden.

Aufgabe 2 (GK-Klausur 1920)



Vierzehn Vierer-Lego-Steine kann man zu einer massiven dreistufigen Pyramide anordnen. Dabei sind insgesamt 36 Noppen zu sehen.

p(n) sei die Anzahl der für eine n-stufige Pyramide benötigten Steine, q(n) die Anzahl der sichtbaren Noppen.

a) Wie viele Steine benötigt man insgesamt für eine 1-, 2-, 3-, 4-, 5-, 6-stufige Pyramide? Fertigen Sie eine Tabelle an.

b) ZP(n) sei der Zuwachs an Steinen gegenüber der vorigen Stufe. Begründen Sie: $ZP(n) = n^2$.

c) Implementieren Sie eine rekursive UND eine iterative Java-Methode `p(int n)` zur Berechnung von p(n).

d) Wie viele Noppen sind insgesamt bei einer 1-, 2-, 3-, 4-, 5-, 6- stufigen Pyramide zu sehen? Fertigen Sie eine Tabelle an.

e) Gegeben sind zwei Funktionen zur Berechnung der Noppenanzahl bei gegebener Pyramidenhöhe n:

```
public static int q1(int n){
  if (n==1) {
    return 4;
  } else {
    return q1(n-1)+8*n-4;
  }
}
```

```
public static int q2(int n){
  if (n==1) {
    return 4;
  } else {
    if (n==2) {
      return 16;
    } else {
      return (q2(n-2) + q2(n-1))/2 + 12*n-10;
    }
  }
}
```

Zeigen Sie anhand des Beispiels n=5 , dass beide Funktionen den gleichen Wert berechnen.

Stellen Sie für die Funktionen jeweils die auftretenden Funktionsaufrufe sowie deren Rückgabewerte graphisch in einem Aufrufbaum dar.

f) Beurteilen Sie die beiden Funktionen bzgl. ihrer Effizienz. Begründen Sie Ihre Antwort.

Aufgabe 3 (Abi-Aufgabe GK)

In der Materialvorlage ist ein Programm gegeben, das alle Möglichkeiten ausgibt, wie ein Faden der ganzzahligen Länge n in Teile der Länge 1 oder in Teile der Länge 2 zerlegt werden kann. Für Teile der Länge 1 wird das Zeichen * ausgegeben und für Teile der Länge 2 wird das Zeichen - ausgegeben.

a) Analysieren Sie den Quelltext und erläutern Sie, wie der Algorithmus arbeitet.

b) (i) Stellen Sie die Prozeduraufrufe für eine Fadenlänge von 5 in einem Baumdiagramm dar.

(ii) Geben Sie an was das Programm für n = 4 ausgibt.

(iii) Geben Sie an:

- wie oft die Methode *zerlege()* für eine Fadenlänge von 5 aufgerufen wird,
- wie oft (für eine Fadenlänge von 5) die Methode *zerlege()* mit n = 3 aufgerufen wird.

c) Implementieren Sie ein vereinfachtes Programm *Faden* und eine Methode *zerlege()* so, dass nur die Anzahl der Möglichkeiten ausgegeben wird.

d) Geben Sie die Anzahl der Möglichkeiten in Form einer Wertetabelle für n = 1 bis n = 8 an.

e) Implementieren Sie eine nicht rekursive Funktion, die die Anzahl der Möglichkeiten einer Zerlegung für einen Faden der Länge n berechnet.

f) Begründen Sie, warum die nicht rekursive Funktion schneller ist als die rekursive Prozedur *zerlege()* in Teilaufgabe c).

Materialvorlage

```java
public class Fadenproblem {

	public static int zeilenNr;
	public static int laenge;

	public static void zerlege(int n, String zerlegung) {
		if (n > 1) {
			zerlege(n - 1, zerlegung + "*");
			zerlege(n - 2, zerlegung + "-");
		} else if (n == 1) {
			zerlege(n - 1, zerlegung + "*");
		} else {
			zeilenNr++;
			System.out.println(zeilenNr + " " + zerlegung);
		}
	}

	public static void main(String[] args) {
		zeilenNr = 0;
		System.out.print("Fadenlänge: ");
		laenge = Input.readInt();
		zerlege(laenge, "");
	}
}
```