Allgemeine Bemerkung:

Bei der Lösung der **Aufgaben 1** bis **10** dürfen **nur lokale Variablen** verwendet werden. Zur Lösung der **Aufgabe 11** sollen erstmalig auch **globale Variablen** verwendet werden. Bitte beachten Sie die entsprechenden Details in den Aufgabenstellungen.

1. Aufgabe

Gegenstand dieser Aufgabe ist die Funktion **f(n) = n mod 3** für Argumente der natürlichen Zahlen einschließlich der 0.

(a) Geben Sie zunächst eine mathematische Darstellung der Funktion **fMod3** an, die ohne die explizite Modulo-Funktion auskommt und nur die Rekursion verwendet.

(b) Schreiben Sie eine Java-Anwendung, welche die Funktion fMod3 in einer eigenen Methode implementiert und in einer main-Methode anwendet. In der main-Methode soll insbesondere die Eingabe von der Tastatur erfragt werden und die Ausgabe auf dem Bildschirm erfolgen.

2. Aufgabe

Gegenstand dieser Aufgabe ist die Wandlung von natürlichen Zahlen von der Dezimaldarstellung in eine duale Darstellung. Genau wie in der vorhergehenden Aufgabe soll dazu nur die Rekursion verwendet werden. Zusätzlich dürfen die mathematischen Funktionen Division und Rest verwendet werden.

(a) Geben Sie eine Skizze des entsprechenden Algorithmus zur Darstellung der Funktion **printDual**, welche bei der Wandlung von Dezimal- in Dualdarstellung gleichzeitig die entsprechenden Bits hintereinander auf dem Bildschirm ausgibt.

(b) Schreiben Sie eine Java-Anwendung, welche die Funktion **void printDual (int n)** in einer eigenen Methode implementiert und in einer main-Methode anwendet. In der main-Methode soll insbesondere die Eingabe von der Tastatur erfragt werden und die Ausgabe auf dem Bildschirm erfolgen.

3. Aufgabe

In dieser Aufgabe soll eine Methode **int multiplikation (int n, int m)** zur Berechnung der Multiplikation von zwei ganzen Zahlen konstruiert werden.

(a) Zunächst wird nur der positive Fall betrachtet: Beide Zahlen n und m sind 0 oder positiv. In diesem Fall soll die Methode multiplikation nur mit Hilfe der Rekursion und der folgenden Hilfs-Methoden realisiert werden: **nachfolger (int n)**, berechnet die nächste ganze Zahl, **vorgaenger (int n)** die nächst kleinere ganze Zahl und **addition (int n, int m)** liefert den Wert der Addition der beiden Zahlen. Die Eingabe soll von der Tastatur eingelesen werden und auf den Bildschirm ausgegeben werden.

(b) Das Programm aus Teil (a) soll so erweitert werden, dass nun auch negative Zahlen möglich sind. Fügen Sie dazu eine Methode **multi (int n, int m) ein**, welche multiplikation verwendet und dies realisiert.

(c) In einer Erweiterung der Java-Implementierung soll die Eingabe und die Ausgabe von Dateien erfolgen, z.B. Eingabe.txt und Ausgabe.txt, die im aktuellen Arbeitsverzeichnis abgelegt sind bzw. werden. Die entsprechenden Dateien sollen nicht binär sein, also vom Entwickler mit einem normalen Editor erstellbar bzw. lesbar sein.

4. Aufgabe

Programmieren Sie ein Glücksspiel. Ausgehend von einem Startkonto von 10.000 Punkten kann ein Spieler einen beliebigen Teilbetrag davon darauf setzen, dass er eine Zahl zwischen 0 und 9 errät. Falls er gewinnt, erhält er das 9-fache seines Einsatzes, sonst ist der Einsatz verloren. Programmieren Sie ein entsprechendes Java-Programm, welches die Eingaben von der Tastatur einliest und die Ausgaben auf dem Bildschirm liefert. Die zu erratende Zahl kann durch einen in Java verfügbaren Zufallsgenerator gezogen werden. Konsultieren Sie dazu die API der Klasse Math im Package java.lang.

5. Aufgabe

Schreiben Sie ein Java-Programm, mit dem auf einfache Weise Nachrichten verschlüsselt und entschlüsselt werden können. Verwenden Sie dazu den einfachen ROT13-Algorithmus (Rotation 13). Bei der ROT13-Codierung werden die einzelnen Buchstaben im Alphabet um 13 Stellen verschoben. Zur Decodierung passiert das gleiche, da das Alphabet 26 Buchstaben enthält.

Reduzieren Sie im Java-Programm das Alphabet auf 26 (lateinische) Buchstaben und entfernen Sie dazu alle Umlaute und Sonderzeichen der deutschen Sprache. Konsultieren Sie die API der Klasse String im Package java.lang, berücksichtigen Sie auch die Möglichkeit, ein Array von chars zu verwenden. Die Eingabe soll einfache (zusammenhängende) Klartexte einzeln von der Tastatur empfangen, verarbeiten und auf dem Bildschirm ausgeben. Testen Sie, ob Ihr Algorithmus korrekt decodiert. Geben Sie dazu den Schlüsseltext wieder ein und vergleichen Sie den neuen Schlüsseltext mit dem ursprünglichen Klartext.

6. Aufgabe

Schreiben Sie ein Programm, das die Grundrechenarten Addition, Subtraktion, Multiplikation und Division für Fließkommazahlen (Double) ausführen kann. Das Java-Programm soll als Taschenrechner implementiert werden, welches drei Parameter in Infix-Notation erwartet, die jeweils durch mindestens ein Leerzeichen voneinander entfernt sind, etwa in der folgenden Form: **java TestRechner 10.1 + 2**. Zur Eingabe verwenden Sie bitte die parse-Methoden, die von den entsprechenden Hüllklassen der primitiven Datentypen angeboten werden. Die Ausgabe soll auf dem Bildschirm erfolgen.

7. Aufgabe

Die folgende Aufgabe soll Sie in einer ersten Näherung an die Klassen-Konzepte heranführen. Berechnen Sie die Zeit in Tagen, Stunden, Minuten und Sekunden, die seit dem 1.1.1980 um 00:00:00 Uhr vergangen ist. Berücksichtigen Sie dabei, dass die Zeit in Java immer in Relation zu einem Referenzdatum (1.1.1970 00:00:00 Uhr) ausgegeben wird, siehe Kapitel 5.4.

Zur Lösung der Aufgabe konsultieren Sie die API der Klasse GregorianCalendar und Calendar. Generell kann eine Methode **method** einer Klasse **klasse** einfach durch die Verwendung der Punkt-Notation aufgerufen werden: **klasse.method**. Der gregorianische Kalender entspricht dem üblicherweise verwendeten Kalender.

8. Aufgabe

Ein Wort wird Palindrom genannt, wenn es von vorne und hinten gelesen identisch ist. Beispiele sind Anna und Lagerregal. Ein nichtleeres Wort ist also offensichtlich ein Palindrom, wenn der erste und der letzte Buchstabe identisch sind und der verbleibende Mittelteil wieder ein Palindrom ist.

Auf der Basis dieser Eigenschaft soll ein Java-Programm implementiert werden, welches die Eigenschaft „Palindrom zu sein" mit Hilfe der Rekursion entscheidet. Konkret soll also eine rekursive Java-Methode der Form **boolean palindrom (String text)** implementiert werden, welche in der main-Methode getestet werden kann. In der main-Methode sollen beliebige Eingabe von der Tastatur gelesen und eine entsprechende Meldung zur Eigenschaft Palindrom auf dem Bildschirm ausgegeben werden.

9. Aufgabe

Gegenstand dieser Aufgabe ist der algorithmische Umgang mit Arrays. In einer Eingabe-Datei ist ein eindimensionales Array gegeben und zwar in nicht binärer Form. D.h. die entsprechenden Werte sind Zeichen. Daher kann die Datei mit Hilfe eines üblichen Editors erzeugt bzw. gelesen werden. Der Name der einzulesenden Datei, welche das Array in Zeichenform enthält, soll von der Kommandozeile eingelesen werden. Ihr Programm soll auf einen fehlenden Namen (Parameter) mit einem Hinweis zum korrekten Aufruf reagieren.

Der üblichen Notation entsprechend wird in der Eingabe-Datei zu Beginn die Dimension des Arrays (Länge) angegeben, gefolgt von den entsprechenden Werten. Die Längen-Angabe erfolgt als int-Wert. Die eigentlichen Datenwerte sind double-Werte.

(a) Schreiben Sie ein Java-Programm, welches die Array-Werte von der in der Kommandozeile angegebenen Datei einliest und in Form eines JAVA-Arrays zurückgibt. Verwenden Sie dazu die Methode **static double [ ] einlesen (String datei)**. Desweiteren soll mit Hilfe der Methode **static double [ ] minMax (double [ ] feld)** das minimale und maximale Eingabe-Array-Element gefunden und in Form eines (Ausgabe-) Arrays zurückgegeben werden. Die Methode **static void ausgabe (double [ ] result)** erwartet die beiden gefundenen Werte in Form eines double-Arrays und gibt diese auf dem Bildschirm aus. In der main-Methode müssen dann nur noch die beschriebenen Methoden nacheinander aufgerufen werden.
Die Kommunikation zwischen den Methoden soll also ausschließlich mit Hilfe von Parametern/Returnwerten erfolgen. Im Wesentlichen sollen zwei Arrays angelegt und mit Hilfe von Referenzvariablen korrekt übergeben werden: Ein Eingabe-Array, welches alle double-Werte aus der Eingabe-Datei enthält und ein Ausgabe-Array, welches die beiden gesuchten Werte Min und Max enthält.

(b) Verändern Sie die Lösung aus Teilaufgabe (a) wie folgt. Die Ein- und Ausgabe soll nun mit Hilfe binärer Dateien erfolgen. Konkret soll die Eingabe von einer binären Datei, die Ausgabe in eine binäre Datei erfolgen. Die Namen der Dateien werden beim Aufruf übergeben. Schreiben Sie dazu zwei separate Programme (Zwei Klassen mit jeweils einer main-Methode), die zum einen die Eingabe-Datei erzeugen (wobei die Werte von der Tastatur erfragt werden) und zum anderen die Werte einer Datei auf dem Bildschirm zwecks Überprüfung ausgeben.

Prof. Dr. M. Duque-Antón

10. Aufgabe

Gegenstand dieser Aufgabe ist die Matrix-Multiplikation (lineare Algebra), siehe dazu auch in Kapitel 5.2 das entsprechende Beispiel im Abschnitt zum Top-Down-Entwurf. Die Aufgabe lautet, eine Matrix M mit einem Vektor V zu multiplizieren und das Ergebnis in eine Datei zu schreiben. Die Eingabe-Matrix bzw. der Eingabe-Vektor V sollen aus entsprechenden Dateien gelesen werden. Bitte beachten Sie, dass diese Operation nicht kommutativ ist. Der Aufruf in Java sollte wie folgt aussehen: **java TestMultiMatrix einVec.txt einMat.txt ausgabe.txt**.

Zur Vereinfachung werden nur int-Werte verwendet. Genau wie in Aufgabe 9 sollen nicht-binäre Dateien verwendet werden. Die Datei enthält also nur Zeichen, die entsprechend interpretiert werden. Zum Aufbau der Eingabe-Dateien verwenden Sie das folgende Prinzip: Die Datei, welchen den Vektor beschreibt, enthält zunächst die Dimension (Länge) gefolgt von der entsprechenden Anzahl an Werten. Die Datei, welche die Matrix definiert, enthält zu Beginn zwei int-Werte, die Anzahl der Zeilen und die Anzahl der Spalten gefolgt von den eigentlichen Werten, die zeilenweise aufgeführt werden. Die Matrix, die den Ergebnis-Vektor aufnehmen soll enthält nur die elementaren Werte (ohne vorherige Dimensions-Angabe), wobei pro Zeile ein Wert stehen soll.

Die Lösung sollte sich an den folgenden Signaturen orientieren. Bitte beachten Sie, dass Sie keine globalen Variablen verwenden dürfen:

```
import java.util.Scanner;
import java.io.*;
public class TestMultiMatrix{
  static int[] einlesenVector (String datei) throws IOException {
  }  // einlesenVector
  static int[][] einlesenMatrix (String datei) throws IOException {
  }  // einlesenMatrix
  static boolean  testCompatibility (int[] vect, int[][] mat) {
  }  // testCompability
  static int[]  multiplikation (int[] vect, int[][] mat)  {
 }  // multiplikation
  static void  auslesenVector (int[] vector, String ausDat) throws IOException {
 }  // auslesenVector
  public static void main (String args [])  throws IOException {
 }  // main
} // TestMultiMatrix
```

Die main-Methode soll die Namen der entsprechenden drei Dateien von der Argumentliste holen und für den Fall, dass die Anzahl der Argumenten-Parameter nicht stimmt, mit einer entsprechenden Meldung vorzeitig terminieren.

Bitte berücksichtigen Sie bei der Lösung, dass alle Arrays (oder besser Array-Objekte) vom Heap erzeugt werden, so dass der entsprechende Speicherplatz im Grunde über die gesamte Lebensdauer des Programms gültig und mit Hilfe entsprechender Referenz-Variablen (die mit Hilfe der Arbeits-Methoden übergeben werden) auch sichtbar ist.

11. Aufgabe

Der Schwerpunkt dieser Aufgabe ist die Entwicklung von Software zur Realisierung von Algorithmen und weniger die Programmierung zahlreicher Ein- und Ausgabevarianten. Aus diesem Grund betrachten wir einen gängigen Ansatz zur Berechnung der Sitzverteilung (in Parlamenten) bei Wahlen.

Treten zur Wahl eines Gremiums mehrere Parteien an, ist der proportionale Sitzanteil auf Basis des Stimmenanteils nur in seltenen Fällen ganzzahlig. Daher ist ein Verfahren zur Berechnung einer ganzzahligen Sitzzahl, die jede Partei in dem Gremium erhält, notwendig.

Das d'Hondtsche Verfahren verwendet Höchstzahlen. Dabei teilt man die Zahl der erhaltenen Stimmen einer Partei nacheinander durch eine aufsteigende Folge natürlicher Zahlen 1, 2, 3, usw. Die dabei erhaltenen Bruchzahlen werden als Höchstzahlen bezeichnet. Als Basis dieser Division (Dividend) wird dabei immer dieselbe Ausgangszahl, also die ursprüngliche Stimmenanzahl der Partei herangezogen. Der Dividend bleibt also in jeder Spalte gleich und wird nur durch den sich verändernden Divisor geteilt.

Die Höchstzahlen werden danach absteigend nach ihrer Größe geordnet. Die so ermittelte Reihenfolge gibt die Vergabereihenfolge der Sitze an. Es finden so viele Höchstzahlen Berücksichtigung, wie Sitze im Gremium zu vergeben sind. Im vorliegenden Beispiel sollen 15 Sitze vergeben werden. Die folgende Tabelle gibt die Stimmenverteilung als Basis für die anschließende Sitzverteilung an:

Partei A	Partei B	Partei C	Partei D
349	301	208	142

Die Dynamik des Verfahrens bzw. die resultierende Sitze können am besten mit Hilfe der folgenden Tabelle erläutert werden. Die Einträge in der Tabelle stellen die Höchstzahlen (Anzahl der Stimmen / aktueller Divisor) dar. Die Werte in Klammern entsprechen dabei der Vergabereihenfolge.

Divisor	Partei A	Partei B	Partei C	Partei D
1	**349 (1)**	**301 (2)**	**208 (3)**	**142 (6)**
2	**174,5 (4)**	**150,5 (5)**	**104,0 (8)**	**71 (12)**
3	**116,33 (7)**	**100,33 (9)**	**69,33 (14)**	47,33
4	**87,25 (10)**	**75,25 (11)**	52	35,5
5	**69,8 (13)**	**60,2 (15)**	41,6	28,4
6	58,17	50,17	34,67	23,67

Aus der Tabelle kann also insbesondere die Sitzverteilung ausgelesen werden. Ein weiterer wichtiger Aspekt kann ebenfalls aus der Tabelle gelesen werden, der für Ihre Bearbeitung allerdings keine Rolle spielt. Die letzte bzw. kleinste Höchstzahl, für die eine Partei noch einen Sitz erhält, gibt den Vertretungswert ihrer Sitze an. Der Vertretungswert ist das Verhältnis aus Stimmen- und Sitzanzahl einer Partei. Am stärksten ist die Partei B vertreten, die für einen Sitz nur 60,2 Wähler benötigt. Partei D hingegegen benötigt für jeden Sitz 71 Wähler.

	Partei A	Partei B	Partei C	Partei D
Sitze	5	5	3	2
Vertretungswert	69,8	60,2	69,33	71

Schreiben Sie ein Java-Programm, welches die Anzahl der Parteien, die Gesamtzahl der Sitze und die Anzahl der Stimmen pro Partei von der Tastatur einliest und die Sitzverteilung auf dem Bildschirm ausgibt, also die Anzahl der Sitze pro Partei.

Alle relevanten Datenstrukturen sollen als globale Variablen verwendet werden. Gehen Sie sparsam mit den zusätzlichen Datenstrukturen um, verwenden Sie nur das Nötigste. Es ist nicht nötig, alle Höchstzahlen im voraus zu berechnen. Statt dessen kann immer nur die nächste Spalte pro Partei zum Vergleich herangezogen werden. Die entsprechenden Spalten müssen nicht in einer Datenstruktur vorgehalten werden, sondern können jeweils aus existierenden Datenstrukturen berechnet werden. Die Lösung sollte sich an den folgenden Signaturen orientieren:

```
import java.util.Scanner;
public class TestDeHondt {

        static int anzParteien, anzSitze;  // Anzahl der Parteien bzw. Anzahl der Sitze insgesamt
        static int [] anzStimmenPartei;   // Nimmt die Anzahl der Stimmen pro Partei auf
                                          // Die Länge des Feldes beträgt anzParteien.
        static int [] anzSitzePartei;     // Zählt die Anzahl der Sitze in Tabelle hoch, enthält
                                          // zum Schluss die Sitzverteilung
                                         // Die Länge des Feldes beträgt anzParteien.
        static int [] divisorPartei;      //  Enthält den aktuellen Divisor pro Partei
                                         // Die Länge des Feldes beträgt anzParteien.

        static void einlesen ( )  { ... } // Alle Datenstrukturen bis auf divisorPartei werden
                                          //  in entsprechende Datenstrukturen eingelesen
        static void initialize ( ) { ... } // divisorPartei wird initialisiert
        static int maxIndex () { ... }  // liefert den Index der Partei zurück mit der aktuell
                                          // größten Höchstzahl
        static void berechneSitze () { ... } // verwendet Methode maxIndex um die Sitze mit
                                           // Hilfe der dynamisch berechneten Tabellen-Einträge
                                           // zu ermitteln. Dabei müssen alle relevanten
                                           // Datenstrukturen ausgegeben werden
        static void ausgabe ( )  { ... }  // Liefert die Siztverteilung auf dem Bildschirm
        public static void main (String args [] )   { ... }

} // TestDeHondt
```

Bitte beachten Sie, dass alle globalen Datenstrukturen vom Typ int sind, die Höchstzahlen aber vom Typ double sein sollten, um Rundungsfehler zu vermeiden. Verwenden Sie daher in diesem Zusammenhang den cast-Operator, siehe Kapitel 4.4.