Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Programmiervorkurs
Aufgabensammlung

Name	Seite	Schwierigkeit (1-10)	Freiheit der Aufgabe (0-3)	Inhalte
Ausgabe von Text	5	1	0	Konsolenausgabe+
Ascii Katze	6	2	0	Konsolenausgabe++
Arbeiten mit Variablen	7	1	0	Variablen+
Visitenkarte 1	8	2	0	Ausgabe+
Volumen - Oberflächenberehcnung	9	2	1	Variablen+, Mathe+
Einlesen und Pi Raten	10	2	1	Variablen+, Mathe+
Ascii und Strings	11	2	0	Datentypen+
Menü	12	3	0	BedingteAnweisungen+
Menü 2.0	13	3	0	Schleifen+ BedingteAnweisungen+
Folgen Erkenne	14	3	1	Schleifen+
Würfeln	15	3	1	Schleifen+
Harmonische Reihe	16	4	1	Schleifne+, Mathe +
Ich Packe meinen Koffer	17	5	2	Schleifen+, Arrays+
Kopfrechnen	18	5	1	Mathe+, Schleifen+
Visitenkarte 2.0	19	4	0	Objektorinetierung+

Name	Seite	Schwierigkeit (1-10)	Freiheit der Aufgabe (0-3)	Inhalte
Drei kleine Schweinchen	20	5	1	Objektorintierung+, Methoden+, Übergabeparameter+
Mission Impossible	21	5	0	Objektorintierung+, Methoden+, Übergabeparameter+, Rückgabewerte+
Schiffe Versenken	22	6	3	Arrays++, Objektorientierung+
N-mal ausgeben	23	3	0	Schleifen+
Bäume Zeichnen	24	7	3	Schleifen++, Logik++
Autohaus	25	5	1	Objektorinetierung+
Index & Nullpointer Exception	26	4	1	Exception+
Zufällige Punkte	27	6	1	Arrays++
Monte-Carlo	28	7	3	Logik++, Mathe+, Schleife+
Newton-Verfahren	29	7	2	Rekursion++, Mathe+
Reaktionsfähigkeit	30	10	3	Recherche++
Noten	31	5	1	Arrays+

Name	Seite	Schwierigkeit (1-10)	Freiheit der Aufgabe (0-3)	Inhalte
Matrizenrechnung	32	9	2	Arrays++, Mathe++, Schleifen+
Maximum+ Insertionsort	33	8	2	Arrays+, Schleife+. Logik+
Korrekte Klammern	34	10	2	Arrays+, Datentypen++
Newton-Iterationsverfahren	35	7	2	Rekursion++, Mathe++
Primzahlen	36	6	1	Rekursion++
Rekursive Funktion	37	4	0	Rekursion+
Referenzen	38	6	0	Referenzen+
Fibanocci	39	6	1	Rekursion+, Schleifen+
Pascalsches Dreieck	40	8	2	Rekursion++, Schleifen++
Rekursive Referenzen	41	6	0	Referenzen+, Rekursion+
Hotel	42	8	3	Arrays++

Aufgabe: Ausgabe von Text

GRUNDLAGE

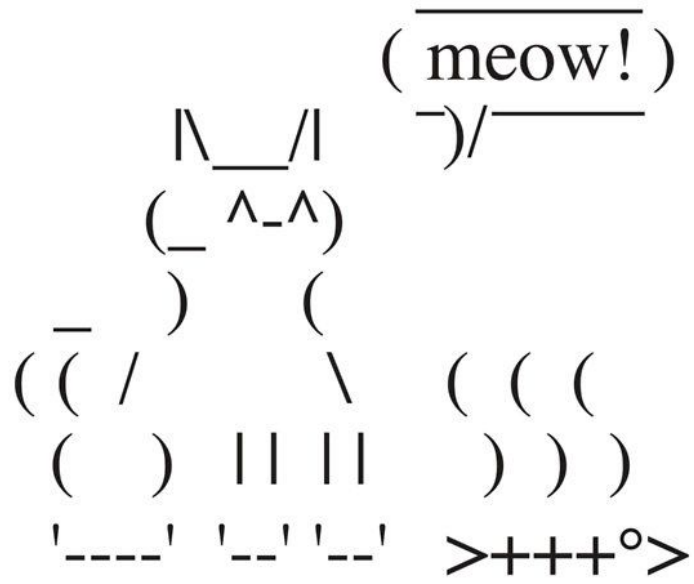
- Schreibt ein Programm mit neuem Projekt, Package und Klasse, das folgende Sätze auf der Konsole ausgibt:
  - Hello World!
  - Hello World!
    How are you?

ZUSATZ

- Schreibe ein Programm, dass den Backslash (\) auf der Konsole ausgibt.

Aufgabe: Ascii Katze

- Lass die Ascii Katze bei dir auf der Konsole erscheinen



Aufgabe: Arbeiten mit Variablen

GRUNDLAGE

- Erstelle ein int i mit Wert 4.
- Addiere 3 dazu.
- Erstelle ein String s mit Wert "7 "
- Nutze die zwei Variablen um auf der Konsole "7 = 7 " auszugeben.

ZUSATZ

- Erstelle ein double mit Wert 1,0
- Erstelle ein double mit Wert 2,5
- Erstelle ein String mit Wert "1,0 + 2,5 = "
- Nutze die 3 Variablen um "1,0 + 2,5 = 3.5 " auf der Konsole auszugeben OHNE außerhalb von System.out zu rechnen und OHNE 3.5 einzugeben.

Aufgabe: Visitenkarte 1

- Schreibt ein Programm, das Eure Visitenkarte ausgibt. Legt dazu Eure Daten in einzelnen Variablen an.
- Tipp: Überlegt Euch vorher, wie viele Variablen Ihr braucht, welchen Datentyp sie haben und wie Ihr sie benennt!
- Ausgabe-Beispiel:
  - Name: Erika Mustermann
  - Straße: Musterstraße 13
  - Ort: 12345 Musterstadt
  - Telefonnummer: 0123 456789
  - Geburtsdatum: 01.01.1970
  - Alter: 47

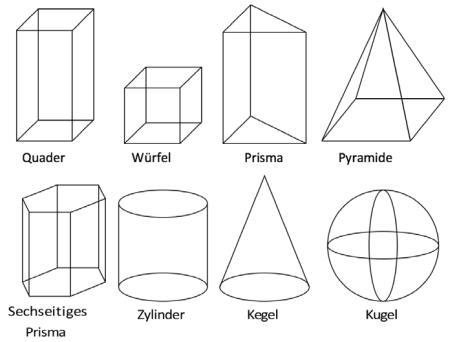
Aufgabe: Volumen- und Oberflächenberechnung

GRUNDLAGE

Schreibt euch ein Programm, das vom Benutzer einen Radius und eine Höhe einliest. Anschließend soll der Benutzer auswählen können, ob das Volumen oder der Oberflächeninhalt des Kegels berechnet und ausgegeben wird.

ZUSATZ

Erweitert das Programm für beliebige dreidimensionale Körper. Fragt den Benutzer dazu am Anfang, für welchen Körper die Berechnungen angestellt werden sollen.



https://www.schuelerhilfe.de/online-lernen/1-mathematik/2-klasse-5/16-k%C3%B6rper/441-geometrische-k%C3%B6rper

Aufgabe: Einlesen verschiedener Datentypen und PI raten

GRUNDLAGE

Schreibt euch ein Programm, das es dem Benutzer ermöglicht einen beliebigen, primitiven Datentyp einzulesen.

ZUSATZ

Schreibt euch ein Programm, das den Benutzer zunächst fragt, auf wie viele signifikante Stellen dieser glaubt PI zu kennen. Anschließend soll der Benutzer bis zu eben dieser Stelle PI eingeben und das Programm überprüfen, ob das Ergebnis richtig ist.

Aufgabe: ASCII und Strings

GRUNDLAGE

Schreibt ein Programm, das vom Benutzer ein Wort einliest und jedes Zeichen, in ASCII-Codierung (dez) umwandelt und auf der Konsole ausgibt.

ZUSATZ

Schreibt ein Programm, das vom Benutzer ein Wort einliest und die Groß- und Kleinschreibung der einzelnen Zeichen vertauscht.

Aufgabe: Menü

› GRUNDLAGE

Ziel ist es, ein Menü zu programmieren, das drei verschiedene Funktionen bietet. Die Funktionen seht ihr unten in der Beispielausgabe. Eine Menüoption soll durch Eingabe einer Kennzahl ausgewählt werden können.

```
1. Hallo ausgeben
2. Zahl quadrieren
3. Zahl größer 0 testen
4. "Programmende" ausgeben

Programmfunktion? 3
Welche Zahl soll geprüft werden? 43
Zahl ist größer 0!
```

Schreibt das Menü 2.0 so um, dass die Fallunterscheidung in einer Methode erfolgt.
Ruft diese fünf mal von main aus auf. Lest dabei jedes mal eine neue Eingabe ein.

Aufgabe: Menü 2.0

- Grundlage

Schreibt die Abfrage nach der gewünschten Menüoption nun in eine Switch/Case-Fallunterscheidung um.

```
switch (eingabe) {
  case 1:
    // "Hallo" implementieren
    break;
  case 2:
    // "Quadrieren" implementieren
    break;
  case 3:
    // "Groesser als" implementieren
    break;
  default:
    // Ende
 }
```

- Schreibt das Menü 2.0 so um, dass die Fallunterscheidung in einer Methode erfolgt.
  Ruft diese fünf mal von main aus auf. Lest dabei jedes mal eine neue Eingabe ein.

Aufgabe: Folgen erkennen

GRUNDLAGE

- Versucht folgende Folgen durch Formulierung einer Schleife auf der Konsole auszugeben:
  - -5 -2 1 4 7 10 13 16 19
  - 3 4 6 10 18 34 66 130 258 514 1026 2050 4098
  - 1 2 4 7 11 16 22 29 37 46 56 67 79 92
- Hinweis: Überlegt euch erst welche Systematik hinter den Folgen steckt und fangt dann an zu programmieren.

ZUSATZ

Schreibt ein Programm, das die folgende Zahlenfolge mithilfe einer Schleife auf die Konsole ausgibt:
1, 3, 5, 7, 9, 11, 13, 15, 17, 19

Aufgabe: Würfeln

Implementiert ein Würfelspiel, bei dem mit zwei Würfeln (1-6) gewürfelt wird. Zu Beginn des Spieles legt der Spieler einen Einsatz (Geldbetrag) fest. Je nach Augensumme der Würfel bekommt der Spieler eine Auszahlung, die der folgende Tabelle entnommen werden kann.

Augensumme	Auszahlung
12	4facher Einsatz
11	3facher Einsatz
10	2facher Einsatz
7, 8, 9	1facher Einsatz
2, 3, 4, 5, 6	keine

Lösungshinweis (Zufallszahlen generieren):

```
Random rnd = new Random();
int wuerfel1 = rnd.nextInt(6);
int wuerfel2 = rnd.nextInt(6);
```

Was muss geändert werden?

Gebt dabei sowohl Augensumme, Auszahlung und Gewinn an.

Aufgabe: Harmonische Reihe mal anders

› Schreibt euch ein Programm, mit dessen Hilfe ihr die harmonische Reihe berechnen könnt:

$$H_n = \sum_{i=1}^{n} \frac{1}{i^{\alpha}}$$ (üblicherweise mit α=1)

› Es soll allerdings eine spezielle Implementierung vorgenommen werden,

- die alle $\alpha \geq 1$ erlaubt.
- die alle $\alpha < 1$ auf 1 setzt.
- für $n < 1$ als Ergebnis 0 zurückgibt.

Aufgabe: Ich packe meinen Koffer

GRUNDLAGE

Schreibt euch ein Programm, welches das Spiel „Ich packe meinen Koffer“ mit zwei Spielern, die gegeneinander spielen, umsetzt. Das Spiel soll zu Ende sein, sobald ein Spieler die Reihenfolge nicht mehr fehlerfrei aufsagen kann.

ZUSATZ

Erweitert das Spiel, sodass nun beliebig viele Spieler spielen können. Vergesst nicht, dass sobald ein Spieler einen Fehler macht dieser in der nächsten Runde bei der Abfrage übergangen wird.

Aufgabe: Kopfrechnen

GRUNDLAGE

Schreibt euch ein Programm, das den Benutzer im Kopfrechnen trainiert mit +- Aufgaben. Dabei sollen sowohl die Zahlen die verrechnet werden, als auch das Ergebnis im Bereich zwischen 0 und 100 liegen.

ZUSATZ

Erweitert euer Programm zum Kopfrechnen, sodass nun auch zusätzlich noch */ Aufgaben trainiert werden können. Achtet wieder auf den Bereich, in dem die Zahlen liegen und dass das Ergebnis aus der Division von Dividend und Divisor eine Ganzzahl ergibt.

Aufgabe: Visitenkarte 2.0

GRUNDLAGE (PABS)

- Erstellt eine Klasse “Visitenkarte“, welche die folgenden Attribute hat: String name, String street, String plz, String city, String phoneNumber, String birthdate, int age
- Erstellt eine weitere Klasse, die die Werte des Benutzers einliest, dann ein Objekt “Visitenkarte“ anlegt und die Werte darin speichert.
- Erzeugt am Ende eine schöne Ausgabe.

ZUSATZ

- Erstellt eine Klasse AutoQuartettKarte, die die Attribute Name, Länge, Hubraum, Leistung und Höchstgeschwindigkeit besitzt. Verwendet keinen Datentyp zweimal.
- Erzeugt 2 Instanzen der Klasse mit beliebigen Werten und vergleicht diese in Bezug auf ein Attribut eurer Wahl.

```
---------------------------------
Name:                Monika Mustermann
Straße:              Musterstraße 1a
Ort:                 12345 Musterstadt
Telefon:             0123 456789
Geb.-Datum:          01.01.1970
Alter:               45
---------------------------------
```

Aufgabe: Drei kleine Schweinchen

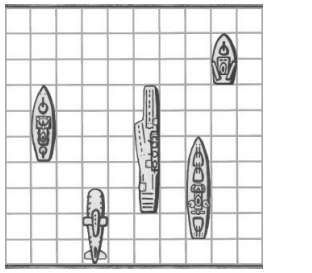
- Erstelle in der Klasse *House* mit den Attribute *owner* und *houseType,* beides Strings. Erstelle eine neue Klasse *Wolf*. Diese hat die Methode *blow(House h)* und das Attribut *name* vom Typ *String*. Der Übergabeparameter ist vom Typ *House*. Die Methode *blow()* gibt jeweils auf der Konsole aus, was passiert, wenn der Wolf pustet und pustet und pustet. Erstelle eine dritte Klasse mit Namen *MainController*, hier kommt die *main*-Methode rein. Erstellt hier drei Objekte vom Typ *House* (jeweils eines mit dem housetype Stroh, Holz und Stein) mit den drei Besitzern *Pfeifer*, *Fiedler* und *Schlau*. Erstelle ein Objekt vom Typ *Wolf*. Übergebt der Methode blow() des Wolfes jeweils die Häuser. Wenn die Methode *blow()* aufgerufen wird, soll nur das Haus aus Stein stehen bleiben.

Link zur Information, falls ihr die Geschichte nicht kennt: https://de.wikipedia.org/wiki/Die_drei_kleinen_Schweinchen_(1933)

Aufgabe: Mission Impossible

- Ein Safe kann nur geöffnet werden wenn von einer Gaunerbande (bestehend aus dem Boss, dem IT-Experten und dem SafeOpener (hardware experte)) jeder seine Aufgabe erfolgreich abschließt.
- Erstellt die Klassen Boss, *SafeOpener*, *ItExpert* und *Safe und SafeControiller. SafeController* beinhaltet die *main*-Methode. *SafeOpener*, *ItExpert* und *Boss* haben jeweils die Attribute *name* (3 zufällige Namen). *SafeOpener* hat die Methode *crackVaultDoorOpen()*, *ItExpert* die Methode *hackSecurity()* und *Boss openSave()*. Jede der Methoden gibt einen *String* mit „*Erfolg*“ oder „*Misserfolg*“ zurück, jeweils mit einer Wahrscheinlichkeit von 50%. Alternativ dürft ihr das mit Booleans lösen.
- *Safe* enthält Methode *getOpenedBy()*, die drei Parameter *String* übernimmt und nur „*Offen*“ ausgibt, wenn alle drei Parameter „*Erfolg*“ beinhalten, sonst wird “*Alarm*“ ausgegeben.
- Der SafeController Initialisiert jeweils eine Instanz der Klassen und startet einen Versuch zum Öffnen des Safes.

Aufgabe: Schiffe versenken



Schreibt euch ein Programm, welches das Spiel „Schiffe versenken“, mit zwei Spielern, umsetzt.
Legt euch dazu zunächst eine Klasse Player an. Diese soll zwei private Attribute haben: den Namen des Spielers und ein Spielfeld mit 8 x 6 Feldern. Schreibt euch zusätzlich Getter- und Setter-Methoden für beide Attribute.
Erstellt euch nun zwei Objekte der Klasse Player, lest von den Nutzern jeweils ihren Namen und 8 kleine Boote ein und spielt nun solange bis ein Spieler keine Boote mehr auf seinem Spielfeld hat (schreibt euch für die Abfrage eine Methode: `public boolean shipLeft()`).

Schreibt euch eine Methode `public static boolean kiPlays(Player p)`. Diese soll eine einfache KI simulieren, die lediglich auf ein zufälliges Feld des übergebenen Spielers p setzt und dann zurückgibt, ob sie getroffen hat oder nicht.

Zusatz: Erweitert Das Spiel um ein Menü, Spielmechanismus und optimiert eure KI.

Aufgabe: N-mal ausgeben

GRUNDLAGE

- Erstellt ein Programm, dass vom Nutzer ein beliebiges Zeichen einliest sowie eine Ganzzahl n. Das Zeichen soll n Zeilen oft n-mal ausgegeben werden.

Aufgabe: Bäume-zeichnen

GRUNDLAGE

- Erstellt ein Programm, das (mit Hilfe von for-Schleifen) symmetrische Tannenbäume (variabler Höhe) generiert. Der Stamm hat immer eine Höhe von zwei Zeilen und die Krone mindestens drei.

```
  *
 ***
*****
  *
  *
```
Höhe 3

```
    *
   ***
  *****
 *******
*********
    *
    *
```
Höhe 5

ZUSATZ

- Zufällige Höhe der Bäume
- Mehrere Bäume nebeneinander gleicher Höhe
- Zufällig viele Bäume zufälliger Höhe nebeneinander

Aufgabe: Autohaus

- Erstellt eine Klasse “Auto“, welche die folgenden Attribute hat:
  String modell, String farbe, int alter
- Erstellt eine weitere Klasse “Autohaus“, die ein Objekt vom Typ “Auto“ anlegt, anschließend die Werte des Autos einliest und in dem Objekt speichert.
- Der Benutzer wird nach jedem Auto gefragt, ob er noch ein weiteres Auto einlesen möchte. Falls “ja“, wird von vorne mit Einlesen begonnen
- Bei der Eingabe des nächsten Autos wird das vorherige Auto durch Neuanlage des Objektes überschrieben (einen “Speicher“ für mehrere Autos implementieren wir später)
- Zum Schluss soll das aktuelle Auto ausgegeben werden

Aufgabe: Index- & Null-Probleme

GRUNDLAGE

- In einer main Methode: Erzeuge ein Array der Größe 1.
- Versuche auf Position 2 zuzugreifen.
- Das Programm wird nun abstürzten. Verhindere dies mit einem try-catch Block.

ZUSATZ

- Erweitere das Programm. Erstelle ein Objekt vom Typ Random innerhalb des try-catch Blocks.
- Setze es danach auf null. (= null nicht = 0)
- Versuche danach mit dem Objekt eine zufällige Zahl zu erhalten.
- Erweitere die try-catch Anweisung so, dass das Programm nicht abstürzt und in jedem Fall “Hello World“ am Ende ausgibt.

Aufgabe: Zufällige Punkte

GRUNDLAGE

- Erstellt ein Programm, das ein dreidimensionales Koordinatensystem durch ein 11x11x11 boolean Array repräsentiert. (Bei 5/5/5 ist der Ursprung.)
- Füllt es dann mit “Punkten“ (= true Einträge), wobei jede Position im Array die Chance von 10% hat, dass dort ein Punkt liegt.
- Gebt die Koordinaten aller Punkte aus.
- Gebt den Punkt aus, der am weitesten vom Ursprung entfernt ist.

Aufgabe: Monte-Carlo

- Lest euch in die Monte-Carlo Methode ein. Es ist ein Grafischer Ansatz um Pi herauszufinden.
- https://de.wikipedia.org/wiki/Monte-Carlo-Simulation
- Es wird gibt ein Quadrat mit einem Viertelkreis. Es werden zufällig Punkte in diesem Quadrat gestreut. Das Verhältnis aller Punkte im Quadrat zum Verhältnis der Punkte die im Viertelkreis liegen, nähert sich immer mehr dem Verhältnis der Flächen an.

Aufgabe: Newton-Verfahren

- Implementiert das Newton Verfahren zum Annähern bzw Lösen der Wurzel-Funktion
- Löst es einmal Rekursiv und einmal Iterativ

Aufgabe: Reaktionsfähigkeit

Schreibt euch ein Programm, das eure Reaktionsfähigkeit testet. Dazu soll Java eine zufällige Zeit warten bis auf der Konsole "Los!" ausgegeben wird. Ab dem Moment soll die Zeit bis zur Betätigung der Enter-Taste gemessen und anschließend ausgegeben werden.

Aufgabe: Noten

GRUNDLAGE

- Erstellt ein Array mit einer Länge von 5 Zellen
- Fragt vom Benutzer 5 Noten (Schulnoten) ab
- Speichert die 5 Noten im Array
- Berechnet den Notendurchschnitt und gebt diesen aus

ZUSATZ

- Schreibt das Programm so um, dass es eine beliebige Anzahl an Noten verarbeiten kann
- Hinweis: Fragt dazu beim Programmstart die Anzahl der Noten ab

http://www.helles-koepfchen.de/artikel/2490.html

Aufgabe: Matrizenrechnung

Schreibt euch ein Programm, das folgende Matrizenoperationen ausführen kann:

- Addition/Subtraktion zweier Matrizen
- Multiplikation einer Matrix mit einem Skalar
- Multiplikation zweier Matrizen
- Negation einer Matrix

› Erstellt euch für jede Rechenoperation eine Methode und eine die das ausgeben der Matrizen auf der Konsole übernimmt

› Die Methoden dürfen die Übergabematrizen nicht verändern

$$y \downarrow \begin{bmatrix} a_{11} & \cdots & a_{1x} \\ \vdots & \ddots & \vdots \\ a_{y1} & \cdots & a_{yx} \end{bmatrix} \rightarrow$$

```
double value = a[y][x];
```

Aufgabe: Maximum und InsertionSort

GRUNDLAGE

Schreibt euch ein Programm, welches das Maximum aus einem Array findet und anschließend dieses mit der vordersten Zahl (Index 0) vertauscht.

ZUSATZ

Implementiert einen einfachen Sortieralgorithmus (InsertionSort). Dieser iteriert durch das Array und setzt dann jeden Wert an die richtige Stelle, indem er den momentanen Wert mit allen vorherigen Elementen vergleicht.

Aufgabe: Korrekte Klammerung

- Implementiert eine Java-Funktion, die für eine Zeichenkette (oder char-Feld) überprüft, ob die dort enthaltenen runden Klammern den Regeln einer vollständigen Klammerung, wie bei einem Ausdruck, entsprechen. Das heißt: Für jede öffnende Klammer '(' muss eine nachfolgende schließende Klammer ')' existieren und die runden Klammern müssen korrekt verschachtelt sein. Zeichen außer den runden Klammern sollen ignoriert werden.
- Folgendes sind korrekt geklammerte Zeichenketten: "()", "", "(()(a)(()((c))))". Diese nicht: "(()", "a (()()) a)".

Aufgabe: Newton-Iterationsverfahren

Das Newton-Iterationsverfahren wird in der Mathematik zum Lösen von nichtlinearen Gleichungen und Gleichungssystemen verwendet.

Zur Berechnung verwendet man folgende Formel:

$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)}$$

Der Startwert x_0 sollte dabei möglichst nahe an dem Endergebnis sein, sodass das Newton-Verfahren in wenigen Iterationsschritten zu einem Ergebnis kommt.

Hinweis: $f'(a) = \frac{f(a+\Delta a) - f(a)}{\Delta a}$

Aufgabe: Primzahlen

GRUNDLAGE

Schreibt euch ein Programm, welches alle Primzahlen von 1 bis zu einer vorgegebenen Zahl ausgibt. Setzt den Algorithmus rekursiv um.

ZUSATZ

Setzt das ganze nun auch iterativ um.

Aufgabe: Rekursive Funktion implementieren

Gegeben sei folgende rekursiv definierte Funktion f:

$f(n) := 1$, für $n = 1$

$f(n) := f(n-1) + 2n - 1$, für $n > 1$

Implementiert eine rekursive Java-Methode, die f(n) berechnet (ohne Iterationen).

Gebt eine nicht-rekursive Implementierung von f an.

Aufgabe: Referenzen

GRUNDLAGE

- Erstelle eine Klasse Zahl mit dem Attribut “int wert“
- In einer main Methode: Erzeuge ein Objekt der Klasse Zahl mit Wert 1. Erstelle außerdem ein int i mit Wert 1.
- Kopiere das Objekt und den int in jeweils eine neue Variable bzw. Objekt.
- Verändere den Wert der originalen Variable und des Objekts.
- Gib den Wert der neuen Variable und des Objekts aus.

ZUSATZ

- Füge der Klasse “Zahl“ eine Methode hinzu, die eine Kopie von dem Objekt erzeugt, dass sie aufruft.
- Diese Kopie soll jedoch so sein, dass man das Original verändern kann, “ohne die Kopie zu ändern“.
- Überprüfe deine Implementierung in einer main Methode.

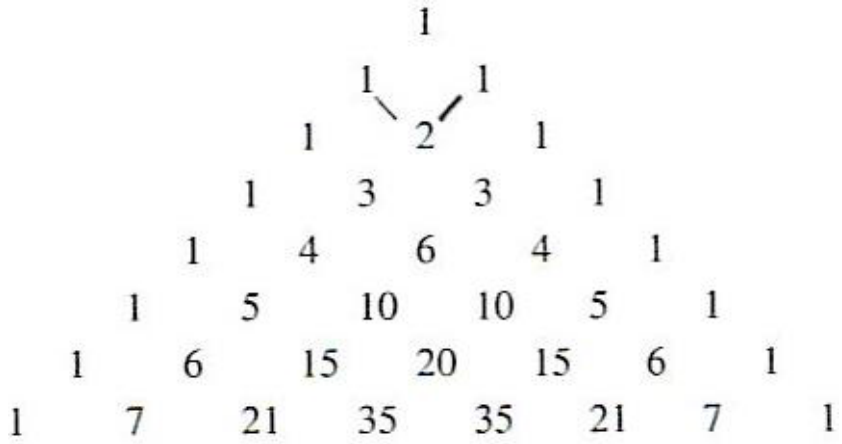
Aufgabe: Fibonacci

GRUNDLAGE

- Schreibt ein Programm, das die Fibonacci-Folge einer Zahl n durch die Methode **`fibRekursiv(int zahl)`** rekursiv berechnet:
  - Fib(n) = 0 //für n=0
  - Fib(n) = 1 //für n=1
  - Fib(n) = Fib(n-1) + Fib(n-2) //Sonst
- Schreibt ein Programm, das die Fakultät einer Zahl n durch die Methode **`fakultaet(int x)`** rekursiv berechnet
  - Fak(n) = 1 //für n=0
  - Fak(n) = Fak(n-1) * n //Sonst

Aufgabe: Pascalsches Dreieck

- Erstellt eine Methode public static int[] pascal(int zeile) die eine Zeile des Pascalschen Dreiecks berechnet. Einmal Iterativ und einmal rekursiv



Aufgabe: Rekursive Referenzen

- In einer main Methode: Erstelle ein Array “feldA“ mit 5 verschiedenen Zahlen.
- Kopiere das Array in zwei neue Variablen:
  - Variable “feldB“ soll auf die gleiche Speicheradresse weisen wie “feldA“.
  - Variable “feldC“ soll den gleichen Inhalt wie “feldA“ haben, aber auf eine unterschiedliche Adresse verweisen.
- Schreibe eine Methode static boolean checkEqual(int[] q, int[] w, int pos), die rekursiv prüft, ob q und w den gleichen Inhalt haben.
  - Hierzu prüft die Methode die Zahlen an <pos>. Sollten sie gleich sein, wird die Methode mit pos+1 aufgerufen.
  - Außerdem müssen die Fälle bedacht werden, wenn der/die Ende(n) der Arrays erreicht werden.
- Verändere in der main Methode den ersten Wert von feldA.
- Rufe in der main Methode die Methode checkEqual mit allen möglichen Kombinationen aus feldA, feldB und feldC auf. Wie viele davon ergeben true?

Aufgabe: Hotel

- Ein Hotel-Komplex hat mehrere Häuser mit jeweils mehrere Etagen die jewils mehrere Zimmer beinhalten.
- Dabei hat nicht jedes Haus gleich viele Etagen und nicht jede Etage gleich viele Zimmer.

- Schreibt ein Verwaltungsprogramm, bei dem zu Beginn der Benutzer seine Parkanlage initialisieren kann (er sagt wie viele Häuser, wie viele Etagen die jeweils haben und wie viele Zimmer die Etagen haben). Danach werden alle Zimmer mit dem Status FREI initialisiert. Der Benutzer soll nun nach Belieben Zimmer mit BELEGT oder REINIGUNG ändern können. Nach einer Belegung muss immer zwingend eine Reinigung erfolgen.

- Tipp: Mehrdimensionale Arrays mit unterschiedlichen Dimensionsgrößen
- Tipp: Ihr Könnt sowohl ein String Arrays anlegen (in dem dann „Belegt“, „Frei“ „Reinigung“ steht, oder aber ein Integer arrays, bei dem 0 für Frei steht etc etc.