Übungen zu Kapitel 8

8.1 BMI:

Schreiben Sie ein Programm, das Ihren ***BodyMassIndex*** berechnet (siehe Übung 5.1). Verwenden Sie das Tool `IO.java` Schreiben Sie eine seperate Methode mit der Schnittstelle:

```
public static double bmi( double gewicht, double groesse )
```

In der Methode soll der BMI aus den Werten der Parameter berechnet und mittels `IO.round()` auf zwei Kommastellen gerundet zurückgegeben werden.

Ermitteln Sie mittels der Methode `IO.promptAndReadString()` den Namen des Verwenders, und lassen Sie eine Begrüssung ausgeben, die den Namen des Verwenders enthält.

Statt Werte für Gewicht und Größe in `main()` "hart" in das Programm zu schreiben, sollen die Werte mittels der Methode `IO.promptAndReadDouble()` eingelesen werden. Rufen Sie die Methode `bmi()` innerhalb von `main()` mit diesen Werten auf.

Gestalten Sie die Ausgaben des Programms ein wenig schöner durch das Einfügen von Leerzeilen mittels `IO.advance()` bzw durch Ausgabe von erläuternden Texten und Unterstreichungen.

8.2 Lineare versus exponentielle Verzinsung:

Ein Programm soll den Unterschied zwischen linearer (kein Zinseszins) und exponentieller Verzinsung (Zinseszinseffekt) ermitteln. Der Benutzer soll mittels `IO.java` das Anfangskapital, den Jahreszinssatz und die Laufzeit der Geldanlage (Anzahl Jahre) eingeben können. Zur Berechnung der Kapitalentwicklung sind zwei Methoden zu entwickeln mit den Schnittstellen:

```
public static double linearZins( double kp, double zs, int n )
public static double expZins( double kp, double zs, int n )
```

Das jeweils erreichte Endkapital inklusive Verzinsung soll auf zwei Kommastellen gerundet an den Aufrufer zurückgegeben werden. Die Methoden sind aus `main()` mit den Eingabewerten aufzurufen. Die Differenz der Endkapitalwerte ist auszugeben.

Anmerkung: Bei linearer Verzinsung wächst das Kapital jedes Jahr um den konstanten Betrag ***Anfangskapital * Jahreszinssatz***. Bei exponentieller Verzinsung wächst das Kapital jedes Jahr um den variablen Betrag ***aktuelles Kapital * Jahreszinssatz***.

8.3 Tilgungsplan mit konstanter Tilgung:

Ein Programm soll einen Tilgungsplan berechnen und ausgeben. Ein Kredit wird mit konstanter jährlicher ***Tilgung*** in `n` Jahren getilgt. Jedes Jahr verringert sich die Restschuld um den Tilgungsbetrag.

Der jährlich aufzubringende Schulddienst heisst ***Annuität*** und enthält neben der Tilgung die anfallenden Schuldzinsen des laufenden Jahres: ***Annuität = Restschuld * zinsSatz + Tilgung***.

Der Benutzer des Programms soll die Kreditsumme, den Jahreszinssatz und den Tilgungszeitraum `n` eingeben können. Die Höhe der Tilgung berechnet sich zu ***Kreditsumme / n***.

Außer `main()` gebe es eine weitere Methode namens `annuitaet()`:

```
public static double annuitaet( double rs, double zs, double tg )
```

Diese wird mit der Restschuld `rs`, dem Zinssatz `zs` und der Tilgungsrate `tg` aufgerufen, berechnet die Annuität und gibt deren Wert an den Aufrufer zurück.

In `main()` werden die erforderlichen Daten vom Benutzer abgefragt. Für jedes verstrichene Jahr sollen die ***Annuität*** sowie die dann ***verbleibende Restschuld*** berechnet und ausgegeben werden. Es muss natürlich solange getilgt werden bis die Restschuld vollständig abgetragen ist.

Berechnen Sie auch, wie gross der ***gesamte Kapitalaufwand*** des Schuldners war; dieser ist gerade der ***Summe*** aller geleisteten Annuitäten.

Alle Konsolenausgaben sollen nur auf zwei Nachkommastellen gerundet erfolgen.

8.4 Fakultät:

Die Fakultät einer nicht-negativen Zahl `n` wird dargestellt durch `n!` und berechnet sich nach folgender Vorschrift:

0! = 1

1! = 1

*2! = 2 * 1!*

*3! = 3 * 2!*

...

allgemein

*n! = n * (n-1)!,* für *n>0*

Implementieren Sie eine iterative und eine rekursive Methode zur Berechnung der Fakultät und erstellen Sie ein Testprogramm für kleine Zahlenwerte. Vorsicht! Die Fakultät wächst sehr rasch, so dass bereits bei `13!` ein `int`-Überlauf stattfindet.

8.5 Fibonacci-Zahlen:

Mit den Fibonacci-Zahlen lassen sich viele Wachstumsprozesse beschreiben. Die Folge dieser Zahlen ist rekursiv definiert durch:

Fib(0) = 1

Fib(1) = 1

Fib(n) = Fib(n-1) + Fib(n-2), für *n>1*

Die nächste Fibonacci-Zahl ergibt sich also aus der Summe der beiden Vorgänger. Implementieren Sie eine rekursive Methode `static int fib_r( int n )`, die die n-te Fibonacci-Zahl berechnet und geben Sie in einem Testprogramm den Wert von *Fib(40)* aus.

Da die Rekursion in diesem Fall sehr ineffizient arbeitet, bietet sich die Verwendung eines iterativen Algorithmus an: Erstellen Sie ein iteratives Pendant `fib_i()` und vergleichen Sie einige Laufzeiten.

Tipp: Man verwendet Variablen `int v=1` und `int vv=1` für den Vorgänger und Vorvorgänger, bildet in einer geigneten Schleife deren Summe und schiebt für die nächste Berechnung `v` und `vv` weiter.

```
public static int fib_i( int n ) {
...
   for( ... ) {
      res = v + vv;
      // für weiteren Durchlauf
      vv = v;  // Vorgänger wird Vorvorgänger
      v = res; // neuer Vorgänger
   }
   return res;
}
```

8.6 Türme von Hanoi:

Die im Kapitel vorgestellte rekursive Methode `bewegeScheiben()` reduziert das Bewegen von ***N***-Scheiben auf das Bewegen von ***N-1*** Scheiben. Was ist hier eigentlich der Basisfall, auf den die Rekursion zuläuft?

Erweitern Sie das Programm Hanoi, so dass die Anzahl der insgesamt benötigten einzelnen Scheibenbewegungen ermittelt und am Ende ausgegeben wird. Verwenden Sie hierzu in der Klasse `Hanoi` eine statische Variable, die jeweils um 1 erhöht wird (etwa nachdem durch `System.out.println()` die Ausgabe erfolgt ist).

Möchte man auf globale Variablen verzichten, so bietet sich die folgende rekursive Alternative an: Wir ändern die Signatur der Methode in

```
public static int bewegeScheiben( int n, char s, char z, char h )
```

und geben die Anzahl der Scheibenbewegungen zurück. Diese berechnet sich aus der Summe der Scheibenbewegungen, die durch die beiden Rekursionsaufrufe verursacht werden plus ***1*** (nämlich das Verschieben der Scheibe *N*):

```
...
s1 = bewegeScheiben( n-1, s, h, z );
System.out.println( n + " von " + s + " nach " + z );
s2 = bewegeScheiben( n-1, h, z, s );
return s1 + 1 + s2;
...
```

Was ist im Fall ***N***=0 zurückzugeben?

Implementieren Sie beide Verfahren und vergleichen Sie die Resultate.