P1 SS24 AI1

Vorbemerkung zum Praktikum

- Grundsätzlich gibt es 4 Aufgabentypen:
  - Vorbereitungs-/Demonstrationsaufgaben **V**x.y: Vorbereitungs-/Demoaufgaben beginnen mit einem V. Die Bearbeitung der Vorbereitungs-/Demoaufgaben ist Pflicht und hat vor Laborbeginn zu erfolgen. Meist wird bei diesen Aufgaben Code gestellt, denn Sie sich anschauen und verstehen sollen. Vermutlich wird dies nicht im Labor kontrolliert. Sie können aber auf jeden Fall Verständnisfragen zu den diesen Aufgaben im Labor stellen.
  - Präsenzaufgaben **P**x.y: Präsenzaufgaben beginnen mit einem P. Die Präsenzaufgaben werden im Labor bzw. während des Labors bearbeitet. Die Lösung der Präsenzaufgaben ist Pflicht. Vermutlich gibt es Präsenzaufgaben nur zum 1.Labortermin.
  - Pflichtaufgaben **A**x.y: Pflichtaufgaben beginnen mit einem A. Die Pflichtaufgaben sind die "normalen" Aufgaben. Wie aus dem Namen klar erkennbar, Pflichtaufgaben sind Pflicht und müssen gelöst werden.
  - Zusatzaufgaben **Z**x.y: Zusatzaufgaben beginnen mit einem Z. Die Bearbeitung der Zusatzaufgaben ist freiwillig. Es ist aber angeraten alle Zusatzaufgaben zu lösen. Vermutlich wird im Labor keine Zeit zum Besprechen der Lösungen von Zusatzaufgaben vorhanden sein. (Falls doch, können auch diese gern besprochen werden).
    U.U. werden Zusatzaufgaben zu einem späteren Zeitpunkt bzw. auf einen nachfolgenden Aufgabenzettel zu Pflichtaufgaben.
- Es gibt zu wenig Vorlesungsvorlauf vor dem ersten Labortermin. In der Vorlesung werden die zur Lösung des 1.Aufgabenzettels benötigten Dinge **nicht** rechtzeitig besprochen worden sein. (Mit dem Vorlesungswissen werden die ersten Staffeln vermutlich nicht einmal die Aufgabenstellung verstehen.) Um den 1.Labortermin sinnvoll nutzen zu können, kommt mit dem 1.Aufgabenzettel eine Sammlung von mehreren kleinen Übungsaufgaben(Zusatzaufgaben) sowie eine Demoaufgabe und eine Präsenzaufgabe (s.o.). Für den 1.Aufgabenzettel gilt, lösen Sie neben D1.1 und P1.1 zum 1.Labortermin diejenigen Zusatzaufgaben, die Sie lösen können (mit möglichen Vorwissen, dass Sie bereits hatten bzw. im Programmierenvorkurs gewonnen haben). Auch wenn dies nicht von den Betreuern kontrolliert werden wird, sollen Sie alle Aufgaben incl. Zusatzaufgaben bis Semesterende lösen. U.U. macht es auch Sinn bereits gelöste Aufgaben zu einem späteren Zeitpunkt mit mehr Wissen noch einmal zu lösen. In der Vergangenheit forderten die Studenten gerade am Anfang immer wieder Aufgaben zum Üben ein. Vor diesem Hintergrund enthält der 1.Aufgabenzettel besonders/extrem viele Aufgaben. Sofern Sie die Aufgaben bis Semesterende nicht lösen können, wäre dies ein deutliches Zeichen, dass Sie die Inhalte der Veranstaltung in einem ungenügenden Maße aufgenommen hätten.
- Sie arbeiten in **Zweier-Teams**. Bitte finden Sie sich schon vor dem 1.Labortermin in Zweier-Teams zusammen.
- Das "echte" Labor beginnt in gewisser Weise mit dem 2.Aufgabenzettel zum 2.Labortermin.
- Zum Erhalt der PVL müssen alle (Pflicht-)Aufgaben gelöst worden sein.
- **Jedes Teammitglied muss** jeden "Teil"/**jede Zeile der vorgestellten Lösung**/Programmcodes **jederzeit erklären und rechtfertigen können.**
- **Jedes Teammitglied muss in der Lage sein,** kurzfristig **kleine Änderungen der Aufgabenstellung implementieren zu können.**
- **Für jede Pflichtaufgabe muss zu Laborbeginn eine Lösung vorliegen.** Nur in seltenen Ausnahmefällen (Sie können die Pflichtaufgabe nicht lösen, sind aber trotzdem gut vorbereitet und insbesondere mit der Aufgabe vertraut) wird toleriert, wenn zum Laborantritt keine Lösung vorliegt.
- Bei den Abnahmen werden seitens der Betreuer nicht immer alle Aufgaben kontrolliert werden. U.U. werden Ihnen Aufgaben "erlassen" im Sinne von: Es gibt kein Abnahmegespräch dazu. Sie müssen aber trotzdem **immer alle** (Pflicht-)**Aufgaben lösen** und alle vorherigen Punkte erfüllen bzw. beachten. U.U. müssen Sie Ihre Lösung auch zu einem nachfolgenden/anderen Termin noch einmal vorführen.
  Weiterhin wird unterschieden in (Pflicht-)Aufgaben beginnend mit einem A und Zusatzaufgaben beginnend mit einem Z. Bei den Zusatzaufgaben ist schon im Vorfeld geplant, dass diese nicht im Labor besprochen werden. Sie sollten eine weitere Möglichkeit zum Üben bieten und es ist ausdrücklich angeraten jede Zusatzaufgabe zu lösen.

Achtung!:
Eclipse hat sehr viele eingebaute Automatismen um erfahrene SW-Entwickler optimal zu unterstützen. Die Erfahrung lehrt, dass dies für Anfänger kontraproduktiv ist. Es ist Ihnen in Ihrem ureigensten Interesse ausdrücklich angeraten für die Lösung der P1-Programmieren-Aufgaben des PTP-Labors nur solche Automatismen zu nutzen, die in der Vorlesung vorgestellt bzw. "freigegeben" wurden.

- **Zur Lösung der Aufgaben des 1.Aufgabenzettels dürfen weder `Arrays` noch `Collections` verwendet werden.**

Aufgabe V1.1 Demos zu den print-Varianten und Variablen

Diese Aufgabe ist eine Vorbereitungs-/Demo-Aufgabe. Auch wenn die Bearbeitung dieser Aufgabe Pflicht ist, wird Ihre Lösung (vermutlich) nicht im Labor kontrolliert. Sie können aber auf jeden Fall Verständnisfragen im Labor stellen.

Im Pub finden Sie die Zip-Datei `v1x1.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt v1x1 lösen.
Schauen Sie sich den gestellten Code bzw. die Klassen `FirstPrintDemo` und `FirstVariableDemo` an.

Zu FirstPrintDemo:
Diese Demo wird durch Anstarten von TestFrameAndStarterForPrintDemo ausgeführt.
Interessant ist der Code im Bereich von Abschnitt1 bis Abschnitt2c. Der Abschnitt 3 ist für Wissbegierige, die dann vermutlich Vorkenntnisse aus der Programmiersprache C haben.
Für die Ausgabe auf den Bildschirm (bzw. in die Eclipse-Console) leisten Ihnen die Methoden **`System.out.print()`**, **`System.out.println()`** und **`System.out.printf()`** nützliche Dienste.
Zwar ist **`printf`** zunächst die Komplizierteste, aber sie ist auch die "Nützlichste". (*Sie ist an das printf() der Programmiersprache C angelehnt*). Die Methode **`printf`** hat als erstes Argument in den runden Klammern einen Formatstring, der von doppelten Hochkommata begrenzt wird. Innerhalb des Formatstrings stehen Formatelemente, welche die Formatierung der Ausgabe regeln. Nach dem Formatstring kommen als weitere Argumente die auszugebenden Variablen bzw. Expressions getrennt durch Komma. Für jeden auszugebenden Wert muss im Formatstring ein Formatelement vorhanden sein. Weitere Informationen gibt es in der API-Dokumentation z.B. unter `java.util.Formatter`. Es ist angeraten im Vorfeld etwas mit printf zu "spielen" um damit vertraut zu werden.

Zu FirstVariableDemo:
Diese Demo wird durch Anstarten von TestFrameAndStarterForVariableDemo ausgeführt. Diese Demo soll zeigen, dass Zustandsvariablen Werte über Methoden Aufrufe hinweg speichern, während die Werte einer lokalen Variable nur innerhalb des jeweiligen Methodenaufrufs verfügbar sind.

Aufgabe P1.1 Bei Einweisung&Vorführung aufpassen und gestellte Aufgabe lösen

Zu Beginn des 1.Labortermins werden sie in unterschiedliche Dinge eingewiesen und danach werden Ihnen einige Dinge gezeigt. Am Ende wird eine Aufgabe gestellt, die Sie lösen müssen.

Hinweis: Die folgenden Zusatzaufgaben: `Z1.1` bis `Z1.8` sind nach "Layout-Gesichtspunkten" angeordnet und nicht nach Schwierigkeitsgrad.

Aufgabe Z1.1 Sportmedaille

Im Pub finden Sie die Zip-Datei `z1x1.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z1x1 lösen.
Nach dem Entpacken finden Sie 2 Klassen vor:

- SportMedalComputer ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen
- TestFrameAndStarter ist eine Möglichkeit Ihre Lösung anzustarten und zu testen.

Schreiben Sie ein Programm zur (Sport-)Wettkampfauswertung. Nach einem Sportwettkampf müssen Medaillen vergeben werden. Junge Teilnehmer (max. 13 Jahre) bekommen

- eine Goldmedaille für 4000 oder mehr Punkte
- eine Silbermedaille für 3000-3999 Punkte
- eine Bronzemedaille für weniger als 3000 Punkte

Ältere Teilnehmer (ab 14 Jahre aufwärts) bekommen

- eine Goldmedaille für 5000 oder mehr Punkte
- eine Silbermedaille für 4000-4999 Punkte
- eine Bronzemedaille für weniger als 4000 Punkte

Berechnen Sie für einen Teilnehmer mit gegebenen Alter (in Jahren) und gegebener Punkteanzahl welche Medaille er/sie bekommt.

Im "markierten Bereich" findet sich die jeweilige Punkeanzahl im Parameter `points` und das jeweilige Alter im Parameter `age`. Das Ergebnis bzw. die jeweilige Medaille soll in der Variablen `medal` abgelegt werden.
"`return medal;` " muss das letzte Statement sein (im "markierten Bereich").

Aufgabe Z1.2 Notenpunkte in Schulnote umrechnen

Im Pub finden Sie die Zip-Datei `z1x2.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z1x2 lösen.
Nach dem Entpacken finden Sie 3 Klassen vor:

- GradeConverter ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen
- TestFrameAndStarter ist eine Möglichkeit Ihre Lösung anzustarten und zu testen.
- Für UnitTestFrameAndStarter siehe einleitenden Text

Früher hatten/bekamen Sie (vermutlich) die Schulnoten 1,2,3,4,5 und 6.
Jetzt haben/bekommen Sie Notenpunkte 15, .., 0.
Rechnen Sie Notenpunkte in Schulnoten um.

3 NP	entsprechen z.B. der Schulnote	"**5+**".
8 NP	entsprechen z.B. der Schulnote	"**3**".
13 NP	entsprechen z.B. der Schulnote	"**1-**".

Im "markierten Bereich" findet sich die jeweilige Anzahl Notenpunkte im Parameter `np`. Das Ergebnis bzw. die jeweilige Schulnote soll in der Variablen `result` abgelegt werden.
"`return result;` " muss das letzte Statement sein (im "markierten Bereich").

Aufgabe Z1.3 Die letzten Drei

Im Pub finden Sie die Zip-Datei `z1x3.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z1x3 lösen.

In der Klasse `LastThree` finden Sie eine Methode `processNewValue()` vor. Beim Aufruf dieser Methode wird eine Zahl "entgegengenommen" (konkret vom Typ `int`) und im formalen Parameter `value` abgelegt. Ihre Aufgabe ist es, sich die letzten 3 Werte zu merken. Beim Aufruf der Methode `printLastThree()` sollen die letzten 3 Werte ausgegeben werden.

Aufgabe Z1.4 Fibonacci–Zahlen

Im Pub finden Sie die Zip-Datei `z1x4.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z1x4 lösen.
Nach dem Entpacken finden Sie 3 Klassen vor:

- FibonacciNumberPrinter ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen
- TestFrameAndStarter ist eine Möglichkeit Ihre Lösung anzustarten und zu testen.
- Für ProposalForAlternativeTestFrameAndStarter siehe einleitenden Text.

Die Fibonacci-Reihe ist eine unendliche Folge von ganzen positiven Zahlen f_0, f_1, f_2, ...
(Siehe http://de.wikipedia.org/wiki/Fibonacci-Folge) Die ersten beiden Zahlen sind:

$f_0 = 0$,
$f_1 = 1$,

Jede weitere Zahl ist die Summe der beiden Vorgängerzahlen :

$f_n = f_{n-2} + f_{n-1}$ für n >= 2

Der Anfang der Fibonacci-Reihe lautet also: 0, 1, 1, 2, 3, 5, 8, ...
Geben Sie die ersten n Werte aus – n soll in einer Variablen **`n`** vom geeigneten Typ abgelegt werden.
Die ausgegeben Zahlen sind durch Komma zu trennen. Vor der ersten Zahl und nach der letzten Zahl darf jedoch **kein** Komma stehen.

Bedenke: Was passiert z.B. bei Eingaben/Startwerten wie 1, 0 oder gar -1 ?

Es sei bemerkt, dass -warum auch immer- die "ursprüngliche" Fibonacci-Reihe mit $f_1 = 1$, $f_2 = 1$, … beginnt.
Es steht Ihnen frei ob Sie die "ursprüngliche" oder die in Wikipedia beschriebene Fibonacci-Folge implementieren. Nur sagen Sie dies vor der Abnahme deutlich an und setzen Sie es konsequent um.

Im "markierten Bereich" findet sich die jeweilige Eingabe bzw. die jeweilige eingeforderte Länge der Fibonacci-Reihe im Parameter `wantedNumberOfFibonacciNumbers`.

Aufgabe Z1.5 Bitanzahl

Im Pub finden Sie die Zip-Datei `z1x5.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z1x5 lösen.
Nach dem Entpacken finden Sie 3 Klassen vor:

- RequiredBitSizeComputer ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen
- TestFrameAndStarter ist eine Möglichkeit Ihre Lösung anzustarten und zu testen.
- Für ProposalForAlternativeTestFrameAndStarter siehe einleitenden Text.
- Für UnitTestFrameAndStarter siehe einleitenden Text

Bestimmen Sie für einen beliebigen positiven[1] ganzzahligen Wert $\mathbf{w} \in \mathbf{N}_0$ vom Typ **`long`**, die Anzahl der Bits, die min. nötig sind um diese Zahl binär zu codieren (unsigned bzw. als "normale" Dualzahl). Oder genauer: Die Anzahl der Bits, die mindestens nötig sind um alle $\mathbf{n} \in \{ z \in \mathbf{N}_0 \mid z \in [0,\mathbf{w}] \}$ binär zu codieren – also auch z.B. kein Zweierkomplement.
Die Berechnung muss korrekt sein - es sind keine Rundungsfehler erlaubt!

Beispiele:

Für den Wert	`0`	lautet das Ergebnis	`1` Bit.
Für den Wert	`1`	lautet das Ergebnis	`1` Bit.
Für den Wert	`2`	lautet das Ergebnis	`2` Bit.
Für den Wert	`255`	lautet das Ergebnis	`8` Bit.
Für den Wert	`256`	lautet das Ergebnis	`9` Bit.
Für den Wert	`140737488355328`	lautet das Ergebnis	`48` Bit.
Für den Wert	`281474976710655`	lautet das Ergebnis	`48` Bit.
Für den Wert	`9223372036854775807`	lautet das Ergebnis	`63` Bit.

Im "markierten Bereich" findet sich die jeweilige Zahl im Parameter `number`. Das Ergebnis bzw. die jeweilige Anzahl benötigter BitsSchulnote soll in der Variablen `resu` abgelegt werden.
"`return resu;` " muss das letzte Statement sein (im "markierten Bereich").

[1] Positiv im für "Programmierer" üblichen Sinn (und mathematisch falsch ;-) also "≥0" bzw. einschließlich 0.

Aufgabe Z1.6 Umrechnung Celsius zu Fahrenheit

Im Pub finden Sie die Zip-Datei `z1x6.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z1x6 lösen.
Nach dem Entpacken finden Sie 2 Klassen vor:

- TemperatureConverter ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen
- TestFrameAndStarter ist eine Möglichkeit Ihre Lösung anzustarten und zu testen.

Erstellen Sie eine 2-spaltige Tabelle, in der in der ersten Spalte die Temperatur in Celsius ausgegeben wird (0 – 100 Grad Celsius, mit der Schrittweite 3 Grad) und in der zweiten Spalte die entsprechende Gradzahl in Fahrenheit.
Zwischen Grad Celsius und Grad Fahrenheit gibt es die Umrechnung:

```
Grad-Fahrenheit = Grad-Celsius * 9/5 + 32.
```

Da mit dieser Aufgabe insbesondere die Effekte des Numeric Promotion *(das Finden eines „gemeinsamen" Types der Operanden für das Ausführen einer Operation)* geübt werden sollen, ist das Ergebnis der Umrechnung in einer ganzzahligen Variablen abzuspeichern, die dann ausgegeben werden soll.
Aus der Aufgabenstellung folgt, dass nur ganzzahlige Werte umgerechnet werden sollen. Celsius ist also immer ganzzahlig.
Die Ausgabe soll wie folgt aussehen:

```
Temperatur-Umrechnungstabelle
=============================
C        F
---------
0       32
3       37
6       43
9       48
12      54
15      59
18      64
21      70
24      75
27      81
30      86
33      91
36      97
39     102
42     108
45     113
48     118
51     124
54     129
57     135
60     140
63     145
66     151
69     156
72     162
75     167
...
```

Achtung!:
Die Ausgabe soll kaufmännisch gerundet (bzw. entsprechend nachfolgender Forderung auf- bzw. abgerundet) sein. D.h. ein Nachkommawert < ,5 wird abgerundet und ein Nachkommawert ≥ ,5 wird aufgerundet
Dies sollen Sie selbst(!) sicherstellen – das Math-Package darf nicht verwendet werden. (Ebenso natürlich andere Packages, die beim Runden unterstützen)

Aufgabe Z1.7 Tannenbaum

Im Pub finden Sie die Zip-Datei `z1x7.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z1x7 lösen.
Nach dem Entpacken finden Sie 3 Klassen vor:

- FirPrinter ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen
- TestFrameAndStarter ist eine Möglichkeit Ihre Lösung anzustarten und zu testen.
- Für ProposalForAlternativeTestFrameAndStarter siehe einleitenden Text.

Ergänzen Sie die Methode **`printFir()`** um Code, der einen Tannenbaum ausgibt (ok, es ist mehr ein Dreieck, aber mit etwas Fantasie, kann man einen Tannenbaum erkennen ;-) Die Ausgabe soll abhängig von einer vorgegebenen Höhe (**`height`**) sein. Sie sollen wie in den nachfolgenden Beispielen Sterne für den Tannenbaum und Punkte für die Luft ausgeben. Um in der Aufgabenstellung nicht bereits schon die Lösung vorzugeben, schauen Sie bitte auf die nachfolgenden Beispiele und entnehmen diesen, die Regeln, wie ein Tannenbaum aussehen soll. Die Farben dienen nur der besseren Lesbarkeit. Ihre Ausgabe soll bezogen auf die Farbe(n) "normal" sein.

Beispiel(e):

Ein Tannenbaum der Höhe 1:

```
*
```

Ein Tannenbaum der Höhe 2:

```
.*.
***
```

Ein Tannenbaum der Höhe 3:

```
..*..
.***.
*****
```

Ein Tannenbaum der Höhe 4:

```
...*...
..***..
.*****.
*******
```

Ein Tannenbaum der Höhe 5:

```
....*....
...***...
..*****..
.*******.
*********
```

Ein Tannenbaum der Höhe 6:

```
.....*.....
....***....
...*****...
..*******..
.*********.
***********
```

Und entsprechend für größere Werte.

Im "markierten Bereich" findet sich die jeweilige Eingabe bzw. die jeweilige Höhe des Tannenbaums im Parameter `height`.

Aufgabe Z1.8 Primfaktorzerlegung

Im Pub finden Sie die Zip-Datei `z1x8.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z1x8 lösen.
Nach dem Entpacken finden Sie 3 Klassen vor:

- PrimeFactorPrinter ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen
- TestFrameAndStarter ist eine Möglichkeit Ihre Lösung anzustarten und zu testen.
- Für ProposalForAlternativeTestFrameAndStarter siehe einleitenden Text.
- Beautician ist nötig für interne Dinge und braucht Sie nicht weiter zu interessieren. (Sie dürfen aber gerne reinschauen und lernen – jedoch ist der Code in einigen Teilen vermutlich für Sie unverständlich)

Berechnen Sie die Primfaktorzerlegung für eine positive Zahl, die in einer Variablen **`zahl`** vom Typ long abgelegt ist. Die Ausgabe soll wie im Beispiel dargestellt aussehen. Also:
<Wert> "=" {<Primfaktor> "*"} <Primfaktor>

Beispiele:

```
                  12 =  2*2*3
                  13 =  13
                  15 =  3*5
                  28 =  2*2*7
                 768 =  2*2*2*2*2*2*2*2*3
          1080664428 =  2*2*3*90055369
         51539607816 =  2*2*2*3*2147483659
         65498029444 =  2*2*37*12689*34877
      37513856612736 =  2*2*2*2*2*2*2*3*3*3*3*3*7*7*7*17*17*23*23*23
     950052134362500 =  2*2*3*3*3*5*5*5*5*5*13*13*23*23*23*37*37
9223372036854775549 =  9223372036854775549
9223372036854775643 =  9223372036854775643
9223372036854775673 =  175934777*52424950849
9223372036854775771 =  19*485440633518672409
9223372036854775777 =  584911*15768846947407
9223372036854775782 =  2*3*3*3*3*17*23*319279*456065899
9223372036854775783 =  9223372036854775783
9223372036854775797 =  3*3074457345618258599
9223372036854775807 =  7*7*73*127*337*92737*649657
```

Achtung: Die Ausgabe schließt mit einem Primfaktor ab und nicht dem Multiplikationszeichen "*"
Bedenke: Was passiert z.B. bei Startwerten wie 2, 1, 0 oder -1 ?

Im "markierten Bereich" findet sich die jeweilige Eingabe bzw. die jeweilige zu zrlegende Zahl im Parameter `number`.

Zusatz-Teil-Aufgabe:

Zerlegen Sie die Zahl 9223372036854775643 in ihre Primfaktoren.
In welches Problem laufen Sie (möglicherweise) dabei? (Dauert es vielleicht etwas lange ;-)
(Nur) für diese (Zusatz-Teil-)Aufgabe dürfen Sie auf *Math.sqrt()* zurückgreifen.
Da die Berechnung einer Wurzel **sehr aufwendig** ist, sollten Sie dies nicht zu oft tun.
Konkret: Für eine Primfaktorzerlegung sollte nur einmal ("am Anfang") eine Wurzel gezogen werden ;-)

Unterstützende Fragen:
Ein Zahl zerfalle in 2 Faktoren f1 und f2 bzw.: **`zahl = f1 * f2`**
Sofern f1 größer der Wurzel ist, was gilt dann für f2? Bzw.: **`f1 > wurzel`** ⇒ …**`f2`**… **?**
Wie lassen sich die zugehörigen "Erkenntnisse" für diese (Zusatz-Teil-)Aufgabe nutzen?
Ist es kritisch, dass **`wurzel`** nur der ganzzahlige Anteil der Wurzel ist?

Syntax-Beispiel:

```
long wurzel = (long)( Math.sqrt(zuZerlegendeZahl) ); // ganzzahlige Anteil der Wurzel
```