Java - Übungsbeispiele
POS1 / Q1

Grundlagen der OOP

Datentypen, Operatoren, Anweisungen, Verzweigungen, Schleifen, Arrays

informatik

Coding-Conventions

Folgende Kodierungsrichtlinien sind beim Programmieren strikt einzuhalten:

- ☐ Naming-Conventions
  - Syntaxregeln für Bezeichner: erlaubte Buchstaben, Ziffern und Sonderzeichen **keine Umlaute!!**
  - Für Bezeichner sind sprechende, **englische** Namen zu verwenden. Ausnahme: für Laufvariablen dürfen einzelne Buchstaben verwendet werden.
  - CamelCase: Großbuchstaben innerhalb eines Wortes
  - Großschreibung für Klassennamen und Interfaces
  - Kleinschreibung für alle anderen Bezeichner
  - Namensgebung für getter- und setter-Methoden: **`getXxx()`**, **`setXxx()`**, **`isXxx()`**
  - Namensgebung für Handler-Methoden: **`onXxx()`**
  - Namensgebung für Controls: **`btEingabe()`**, **`tfAusgabe()`**,...
- ☐ Indentation: ***Shift-Alt-F*** → Netbeans Autoformater
  - **Einrücken !!**
  - Zeilenumbrüche
  - Zeilenlänge max. 80 Zeichen (rote Linie bei Netbeans)
- ☐ Kommentare:
  - JavaDoc-Kommentare für alle selbst implementierten Methoden, mit Ausnahme von Konstruktoren sowie getter- und setter-Methoden
  - Implementierungs-Kommentare
- ☐ Nur eine Anweisung pro Zeile
- ☐ Nur eine Deklaration pro Zeile

Klasse, Objekt, Attribut, Methode

Die Bespiele für dieses Quartal enthalten neben der Einführung in die Grundkonzepte der Objekt-Orientierung die Wiederholungen folgender Inhalte, die aus C bekannt sind: Datentypen, Operatoren, die Anweisungen **`if`**, **`switch`**, **`for`** und **`while`**, sowie Arrays.

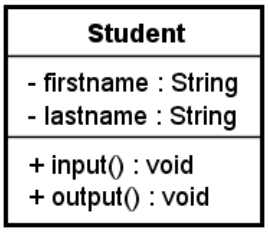
Bsp_101_HelloWorld
Erstelle die Java-Klasse **`HelloWorld`**. In der **`main()`**-Methode wird mit Hilfe der **`println`**-Anweisung **`"Hello World"`** auf der Konsole ausgegeben.

Bsp_102_Person
Erstelle die Klasse **`Person`**. In der **`main()`**-Methode werden mit Hilfe der **`Scanner`**-Klasse ein Vor- und ein Nachname auf zwei String-Variablen eingelesen. Anschließend werden die beiden Werte auf der Konsole ausgegeben:

```
Bitte Vorname eingeben / Please enter your firstname: David
Bitte Nachname eingeben / Please enter your surname: Alaba
------------------------------------------------------------------
Der Name lautet: David Alaba
The name is: David Alaba
```

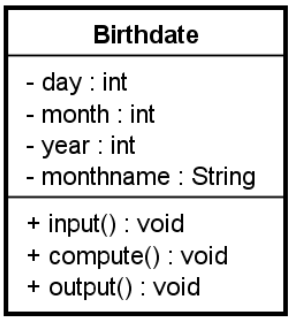
Kompetenzen: Klasse

Bsp_103_Student
Erstelle die Klasse **`Student`** gleich wie die Klasse **`Person`**. Die Klasse **`Student`** wird aber in die Methoden **`input()`**, **`output()`** sowie die Attribute **`firstname`** und **`lastname`** strukturiert. In der **`main()`**-Methode werden zwei Objekte der Klasse **`Student`** erzeugt, und durch Aufruf aller Methoden getestet.



Kompetenzen: Klasse, Objekt, Attribute, Methoden

Bsp_104_Birthdate
Erstelle die Klasse **`Birthdate`** zum Speichern der Geburtsdaten. Die Klasse wird in die Methoden **`input()`**, **`compute()`** und **`output()`** sowie die Attribute **`day`**, **`month`**, **`year`** und **`monthname`** strukturiert. In der **`main()`**-Methode wird ein Objekt der Klasse erzeugt, und durch Aufruf aller Methoden verwendet.



Beispiel:

```
Enter your birthday:
Day: 1
Month: 1
Year: 2003
Your birthday is: 01.January.2003
```

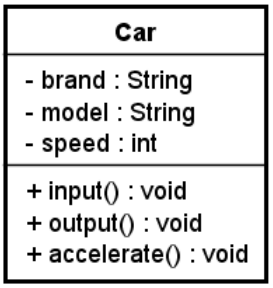
Erweiterungen des Programms:

- In der Methode **`input()`**: für die Eingabe müssen gültige Wert, d.h. Tage von 1-31, Monate 1-12 verwendet werden, sonst wird die Eingabe wiederholt.
- Mit der Methode **`compute()`** wird der Tag des Jahres berechnet und auf einer Instanzvariable **`dayOfYear`** gespeichert. In der Methode **`output()`** wird der Wert ausgegeben.
- Mit der Methode **`compute()`** wird der Wochentag (Monday, … Sunday) berechnet und auf einer Instanzvariable **`dayOfWeek`** gespeichert. Recherchiere im Internet nach einem geeigneten Algorithmus! In der Methode **`output()`** wird der Wert ausgegeben.

Kompetenzen: Klasse, Objekt, *Attribute, Methoden*

Bsp_105_Car

Erstelle die Klasse Car mit den Attributen **`brand`**, **`model`** und **`speed`** sowie den Methoden **`input()`**, **`output`**() und **`accelerate`**(). In der Methode **`input()`** werden die Werte für **`brand`** und **`model`** eingelesen. In der Methode **`accelerate()`** wird die Geschwindigkeit (**`speed`**) um 1 Km/h erhöht. In der Methode **`output()`** werden alle aktuellen Werte ausgegeben.
Erzeuge in der **`main()`**-Methode ein Objekt der Klasse **`Car`**. Lies die gewünschte Geschwindigkeit ein und beschleunige das Auto mit der **`accelerate()`**-Methode so lange bis die gewünschte Geschwindigkeit erreicht wurde.



Beispiel:

```
Enter the brand of the car: VW
Enter the model of the car: Golf
VW Golf runs with 0 Km/h
Desired velocity: 38
VW Golf runs with 38 Km/h
```

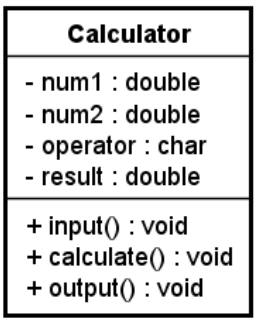
Erweiterungen des Programms:

- In der Methode **`main()`**: die gewünschte Geschwindigkeit muss >0 und <130 Km/h sein, sonst wird die Eingabe wiederholt.
- Mit der Methode **`slowDown()`** wird die Geschwindigkeit um 1 Km/h verringert. Verwende diese Methode im **`main()`** um das Auto auf eine gewünschte Geschwindigkeit abzubremsen.

Kompetenzen: Klasse, Objekt, Attribute, Methoden

Bsp_106_Taschenrechner

Erstelle die Klasse **`Calculator`** die zwei Zahlen und einen Operator einliest, das Ergebnis berechnet und ausgibt. Die Klasse ist in die Methoden **`input()`**, **`calculate()`**, **`output()`** sowie die Attribute **`num1`**, **`num2`**, **`operator`** und **`result`** zu gliedern,
In der **`main()`**-Methode soll die Klasse getestet werden. Verwende in der Methode **`output()`** folgendes Format für die Ausgabe:



Beispiel:

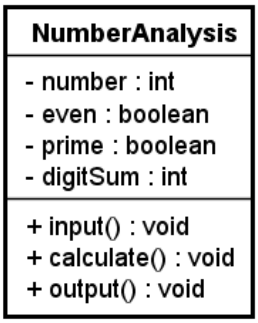
```
Erste Zahl eingeben: 21
Operator eingeben: +
Zweite Zahl eingeben: 5
Addition: 21 + 5 = 26
```

Erweiterungen des Programms:

- In der Methode **`input()`**: Überprüfe den Operator auf Richtigkeit: es dürfen nur die Zeichen +, -, *, / eingegeben werden. Sonst muss die Eingabe wiederholt werden.
- Erstelle in der **`main()`**-Methode ein zweites Objekt der **`Calulator`** Klasse und teste es ebenfalls.

Kompetenzen: Klasse, Objekt, Attribute, Methoden

Bsp_107_Zahlenanalyse



Erstelle die Klasse **`NumberAnalysis`** entsprechend dem Klassendiagramm.
In der Methode **`input()`** wird eine Zahl eingelesen, in **`calculate()`** erfolgt die Zahlenanalyse: ob es sich um eine gerade Zahl handelt (**`even`**), ob es eine Primzahl ist (**`prime`**) sowie die Berechnung der Ziffernsumme (**`digitSum`**). Die Analyse soll auch für negative Zahlen funktionieren. In der **`output`**() Methode werden die Ergebnisse der Analyse ausgegeben.

Beispiel:

```
Zahl eingeben: 123
123 ist ungerade, keine Primzahl, Ziffernsumme: 6
```

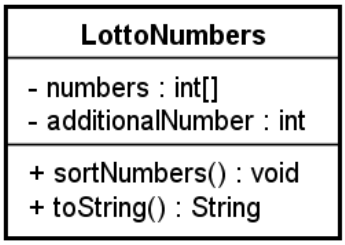
Erweiterungen des Programms:

- Wiederhole in der **`main()`**-Methode die Zahlenanalyse so lange, bis der Benutzer das Programm durch Eingabe von 0 abbricht.

Kompetenzen: Klasse, Objekt, Attribute, Methoden

Constructor, getter, setter

Bsp_201_Lottozahlen



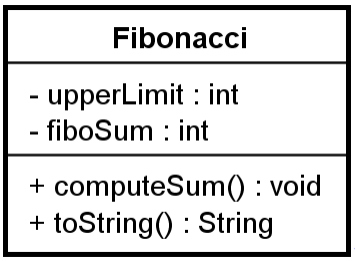
Erstelle die Klasse **`LottoNumbers`** entsprechend dem Klassendiagramm.
Im Konstruktor der Klasse werden 6 verschiedene Lottozahlen im Bereich 1 <= zz <=45 erzeugt und auf dem Feld **`numbers`**. ge speichert. Weiters wird eine Zusatzzahl, die im Feld noch nicht vorkommt, auf der Variable **`additionalNumber`** gespeichert. In der Methode **`sortNumbers()`** werden die Zahlen im Feld aufsteigend sortiert. Die Methode **`toString()`** erzeugt einen String für die Ausgabe der Zahlen.

Beispiel:

```
Lottoziehung: 6,9,14,23,34,37 Zusatzzahl: 7
```

Kompetenzen: Konstructor

Bsp_202_Fibonacci



Erstelle die Klasse **`Fibonacci`** entsprechend dem Klassendiagramm.
Die Fibonacci-Folge wird gebildet durch Addition der beiden vorhergehenden Zahlen der Folge:

```
1,1,2,3,5,8,13,21,34,…
```

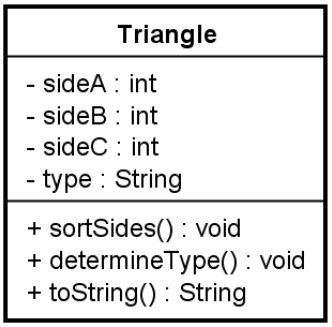
Lies eine ganze Zahl von der Konsole ein und berechne die Summe aller geraden Fibonacci-Zahlen kleiner gleich der eingelesenen Zahl.
Die Eingabe der Zahl erfolgt in der **`main()`**-Methode. Verwende den Konstruktor der Klasse zum Initialisieren der Instanzvariablen. In der Methode **`computeSum()`** wird die Fibonacci-Summe berechnet. Über die **`toString()`**-Methode erfolgt die Ausgabe.

Beispiel:

```
Obere Grenze: 4000000
Fibonacci-Summe bis 4000000: 4613732
```

Kompetenzen: Konstructor

Bsp_203_Dreieck



Erstelle die Klasse **`Triangle`** im Package **`triangle`** entsprechend dem Klassendiagramm.
Die Eingabe der 3 Seitenlängen erfolgt in der **`main()`**-Methode. Verwende den Konstruktor der Klasse zum Initialisieren der Seiten a, b und c. In der Methode **`sortSides()`** werden alle 3 Seiten aufsteigend sortiert.
In der Methode **`determineType()`** wird der Typ des Dreiecks bestimmt:

- gleichseitig (a = b = c)
- gleichschenkelig (a ≠ b = c , a = b ≠ c , a = c ≠ b)
- rechtwinkelig ($a^2 + b^2 = c^2$)
- allgemein (a + b > c)
- kein Dreieck

In der **`toString`**() Methode wird das Ergebnis ausgegeben.

Beispiel:

```
Eingabe Seite A: 5
Eingabe Seite B: 3
Eingabe Seite C: 4
Dreieck (3,4,5) -> rechtwinkelig
```

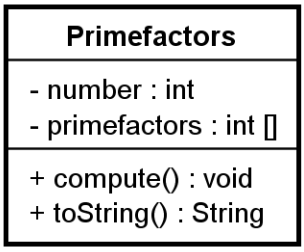
Erweiterungen des Programms:

- Erstelle eine weiteren, parameterlosen Konstruktor, der für die Seiten a, b und c zufällig Werte zwischen 5 und 10 erzeugt. Teste den Konstruktor in der **`main()`**-Methode.

Kompetenzen: Konstructor

Bsp_204_Primfaktoren

Erstelle die Klasse **`Primefactors`** entsprechend dem Klassendiagramm.
Das Einlesen der Zahl, für die die Primfaktoren berechnet werden sollen, erfolgt in der **`main()`**-Methode. Verwende eine setter-Methode zum Initialisieren des Attributs **`number`**.



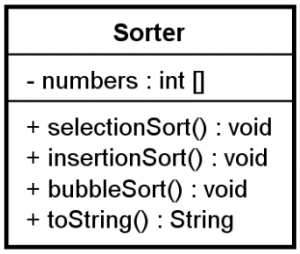
Beispiel:

```
Zahl: 13195
Primfaktoren: 5, 7, 13, 29
```

Kompetenzen: Konstructor, setter-Methoden

Bsp_205_ZahlenSortierer

Erstelle die Klasse **`Sorter`** entsprechend dem Klassendiagramm.
Die Zahlen werden auf einem int-Feld **`numbers`** gespeichert. Die Größe des Feldes wird in der **`main()`**-Methode eingelesen und an den Konstruktor der Klasse **`Sorter`** übergeben. Im Konstruktor wird das Feld mit Zufallszahlen 0 < zz < sizeOfArray initialisiert.
Der Benutzer kann über die Menüauswahl das Sortierverfahren auswählen. Bei der Ausgabe werden nur die ersten 10 und die letzten 10 Zahlen des Feldes ausgegeben. Wenn das Feld weniger als 20 Elemente hat so wird das gesamte Feld ausgegeben.



Beispiel:

```
Anzahl der Elemente: 100
(1)Selection-Sort
(2)Insertion-Sort
(3)Bubble-Sort
Wählen sie aus: 2
1, 1, 2, 3, 5, 5, 6, 7, 8, 9
. . .
91,91,92,92,94,95,96,96,97,98
```

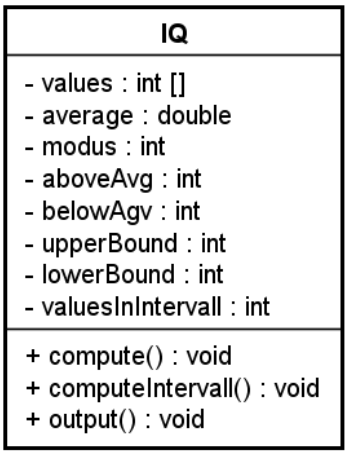
Erweiterungen des Programms:

- Erweitere die Klasse **`Sorter`** um die Methode **`quickSort()`** mit dem rekursiven Quicksort-Verfahren.

- Erweitere die Menüsteuerung in der **`main()`**-Methode sodass die Auswahl wieder von vorne beginnt. Es ist nach jeder Auswahl ein neues Objekt der Klasse **`Sorter`** zu erzeugen.
- Erweitere die **`main()`**-Methode um eine Zeitnehmung, die die Dauer des Sortiervorgangs auf Millisekunden genau misst und ausgibt.

Kompetenzen: Konstruktor

Bsp_206_Intelligenzquotient



Erstelle die Klasse **`IQ`** entsprechend dem Klassendiagramm.
In der **`main()`**-Methode werden alle IQ-Werte, getrennt durch Strichpunkte als ein String eingelesen. Der String wird an den Konstruktor übergeben und dort mit Hilfe eines Scanners (Delimiter ändern!) in einzelne Zahlen konvertiert und auf das Feld **`values`** hinaufgeschrieben.
In der **`compute()`**-Methode werden folgende Berechnungen durchgeführt: das arithmetische Mittel (**`average`**), der Modus (**`modus`**), die Anzahl der Personen mit einem IQ unter bzw. über dem Durchschnitt (**`belowAvg`** und **`aboveAvg`**). In der **`output()`**-Methode erfolgt die Ausgabe.
Nach diesen Berechnungen werden in der **`main()`**-Methode die Grenzen für das Intervall eingelesen und mit **`setter`**-Methoden an die Klasse übergeben. In der Methode **`computeIntervall()`** erfolgt die Berechnung wie viele der IQ Werte sich im Intervall befinden (**`valuesInIntervall`**). Die Ausgabe wiederum erfolgt im **`main()`**, der Zugriff auf die Werte über **`getter`**-Methoden.
Schreibe die **`main()`** Methode in einer eigenen Klasse **`IQTester`**.

Beispiel:

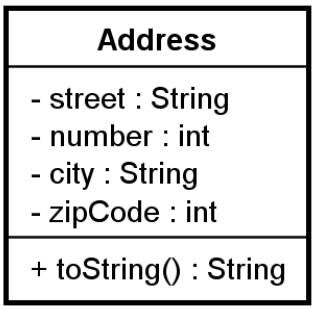
```
Bitte gib die Liste der IQs durch Strichpunkt getrennt ein:
131;101;75;103;110;89;101;90;127;101;94

Auswertung:
Folgende Werte wurden eingegeben:
 131 101 75 103 110 89 101 90 127 101 94
Arithmetisches Mittel: 102,00
Der häufigste Wert (= Modus): 101
Anzahl der Personen mit IQ < 102,00:  7
Anzahl der Personen mit IQ > 102,00:  4
Bitte gib den Bereich ein, um die Anzahl der IQs zu bestimmen, die
im Intervall liegen.
Untergrenze: 110
Obergrenze: 130
Im Bereich [110,130] liegen   2 IQ-Werte
```

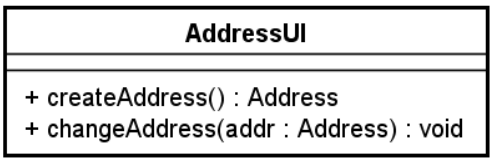
Kompetenzen: Konstruktor, setter-Methoden

2-Klassen-Konzept, Methoden überladen, VarArgs

Bsp_207_AdressenVerwaltung



Erstelle die Klassen **`Address`** und **`AddressUI`** entsprechend dem Klassendiagramm. Das Programm dient zum Einlesen und Ändern einer Adresse.
Die Klasse **`Address`** ist eine reine Datenklasse mit Konstruktor, getter, setter und **`toString()`**-Methode.
In der Klasse **`AddressUI`** wird ein **`Address`**-Objekt eingelesen und erzeugt (Methode **`createAddress()`**) und ein **`Address`**-Objekt geändert (**`changeAddress()`**). Beachte Übergabeparameter und Return-Types!

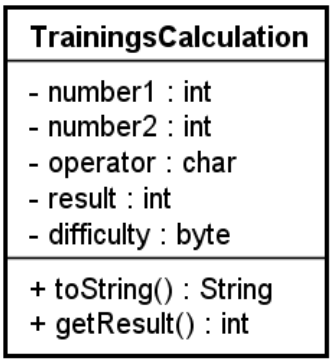


Beispiel:

```
Enter street: Rodeo Drive
Enter number: 123
Enter city: LA
Enter zipcode: 90210
Address: Rodeo Drive 123 - 90210 LA
What would you like to change?
  (A) street: Rodeo Drive
  (B) number: 123
  (C) city: LA
  (D) cipcode: 90210
a
Enter new value: Sunset Blvd
Address: Sunset Blvd 123 - 90210 LA
```

Bsp_208_Rechentrainer

Erstelle die Klassen **`TrainingsCalculation`** und **`CalcTrainerUI`** entsprechend dem Klassendiagramm. Das Ziel des Programms ist es eine Trainingseinheit mit mehreren Mathe-Aufgaben in zwei Schwierigkeitsstufen durchzuführen.

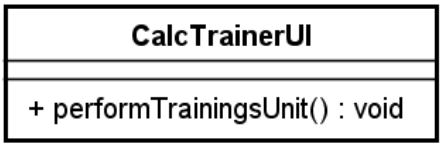


In der Klasse **`TrainingsCalculation`** wird eine Rechnung abgebildet. Im Konstruktor werden alle Instanzvariablen initialisiert. In Schwierigkeitsstufe 1 werden Zufallszahlen zwischen 1 und 10 erzeugt, in Schwierigkeitsstufe 2 zwischen 1 und 100. Der Operator für die 4 Grundrechnungsarten +,-,*,/ wird zufällig erzeugt. Das Ergebnis ist ganzzahlig.
In der **`toString()`** Methode wird die Rechnung als String zurückgegeben.

In der Methode **`performTraingsUnit()`** der Klasse **`CalcTrainerUI`** wird eine Übungseinheit mit mehreren Rechenaufgaben durchgeführt: der User gibt zuerst die Anzahl der Runden und die Schwierigkeit ein. Für jede Runde erfolgt:

- die Ausgabe der Rechnung vom Programm
- die Eingabe vom Ergebnis durch den User
- die Ausgabe ob das Ergebnis richtig war oder nicht

Am Ende wird die Anzahl an erreichten Punkten ausgegeben.



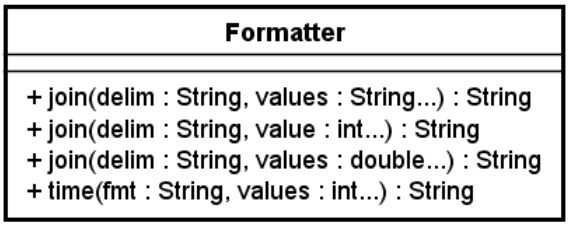
Beispiel:

```
Enter number of rounds: 3
Enter difficulty: 1
Round 1
  9 - 4 = ?
  Enter result: 5
  perfect!
Round 2
  3 * 1 = ?
  Enter result: 4
  The correct result is 3
Round 3
  8 + 3 = ?
  Enter result: 11
  perfect!
you achieved 2 of 3 points
```

Kompetenzen: Konstruktor, getter-Methoden, 2-Klassen Konzept

Bsp_209_Formatter

Erstelle die Klassen **`Formatter`** entsprechend dem Klassendiagramm.



Die Methode **`join()`** dient zum Verketten von Zahlen oder Strings mit einem Delimiter. Die Werte werden an die Methode übergeben, zurückgegeben wird ein String der die verketteten Werte enthält. Double-Werte werden auf 2 Kommastellen genau formatiert.

Die Methode **`time()`** dient zum Formatieren einer Uhrzeit, gegeben durch die Anzahl an Stunden, Minuten und ev. Sekunden. Der Rückgabewert ist der formatierte Uhrzeit-String.
Folgende Formate werden unterstützt:
`hh:mm:ss` → **`14:20:30`** → Ausgabe von Stunden (im 24 Stunden Format), Minuten und Sekunden
`HH:mm:ss` → **`02:20:30 PM`** → Ausgabe von Stunden (im 12 Stunden Format), Minuten und Sekunden
`hh:mm` → **`07:20`** → Ausgabe von Stunden (im 24 Stunden Format) und Minuten
`HH:mm` → **`07:20 AM`** → Ausgabe von Stunden (im 12 Stunden Format) und Minuten
Wird an die Methode ein ungültiger Format-String oder eine ungültige Anzahl an Parametern übergeben, so wird ein leerer String zurückgegeben.

In der Klasse **`FormatTester`** werden alle Methoden der **`Formatter`**-Klasse mit zumindest 10 unterschiedlichen Testfällen (ohne Benutzereingabe) getestet.

Beispiele für **`join()`**:
```
Montag-Dienstag-Mittwoch-Donnerstag-Freitag
1,2,3,4
1
0,50 - 0,33 - 0,25
```
Beispiele für **`time()`**:
```
12:20:30
10:45
06:25 PM
11:33:00 AM
```

Kompetenzen: VarArgs, Methoden überladen

Erweiterungen des Programms:
- Erweitere die Klasse **`Formatter`** um 3 weitere, überladene Varianten der Methode **`join()`** die den Delimiter als **`char`** und nicht als **`String`** übergeben bekommen.
- Erweitere die Klasse **`Formatter`** um die Methode **`date(),`** die ein formatiertes Datum, in unterschiedlichsten Formen, zurückgibt.

Statische Methoden und Variablen

Bsp_301_MyMath ☆

Erstelle die Klasse **MyMath** die folgende statische Variablen und Methoden zur Verfügung stellt:

double PI : Wert der Zahl π als Konstante auf 15 Kommastellen genau (aus dem Internet).
int abs(int a): Gibt den Absolutbetrag der Zahl ***a*** zurück.
int min(int a, int b, int c): gibt den kleinsten Wert der Zahlen ***a***, ***b*** und ***c*** zurück.
int max(int a, int b, int c): gibt den größten Wert der Zahlen ***a***, ***b*** und ***c*** zurück.
double power(double basis, int exp): gibt den Wert von ***basis*** hoch ***exp*** als double-Wert zurück. Soll auch für negative Werte von Basis und Exponent funktionieren.
double round(double wert, int stellen): gibt die zahl ***wert*** gerundet auf die durch ***stellen*** definierte Anzahl an Kommastellen zurück.
long fakultaet(int zahl): gibt die Fakultät von ***zahl*** zurück: n! = n* (n-1)* … *3*2*1 mit 0! = 1
double sin(double x): gibt den Sinus von ***x*** zurück.
double cos(double x): gibt den Cosinus von ***x*** zurück.

Alle Berechnungen sind ohne Verwendung der API-Klasse **Math** durchzuführen.
Zur Berechnung von Sinus und Cosinus werden folgende Exponentialfunktionen verwendet:

$$\sin(z) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot z^{2\cdot k+1}}{(2\cdot k+1)!} = z - \frac{z^3}{3!} + \frac{z^5}{5!} - \frac{z^7}{7!} \pm \cdots ,$$

$$\cos(z) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot z^{2\cdot k}}{(2\cdot k)!} = 1 - \frac{z^2}{2!} + \frac{z^4}{4!} - \frac{z^6}{6!} \pm \cdots .$$

Verwende jeweils die ersten 10 Glieder für die Berechnung und benutze die bereits implementierten Methoden **power()** und **fakutaet()**.

Erstelle die Klasse **MathTesterGUI** in der Textfelder für die Ein- und Ausgabe, sowie Buttons für die verschiedenen Berechnungen zur Verfügung stehen.

Folgende Testfälle sind zumindest zu prüfen:

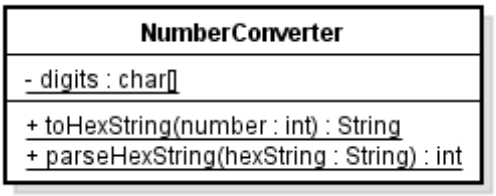
- Eingabe einer negativen Zahl und Ausgabe des Absolutbetrags dieser Zahl
- Eingabe von drei Zahlen und Ausgabe der größten und der kleinsten dieser Zahlen
- Ausgabe der Zahl π
- Ausgabe der Zahl π auf 6 Kommastellen gerundet: 3.1415930000000
- Ausgabe der Fakultät einer eingelesenen Zahl
- Ausgabe von Sinus und Cosinus für die Werte: 0, π/4, π/2 und π
- Ausgabe von Sinus und Cosinus für einen eingelesenen Wert
- Vergleiche die Ergebnisse der Berechnungen von Fakultät, Sinus und Cosinus mit Internet- oder Taschenrechner-Ergebnissen

Kompetenzen: Wiederholung: Schleifen
Neu: static
Umfang: 1 DP-Std

Bsp_302_NumberConverter ☆

Erstelle die Klasse **`NumberConverterBL`**, die zwei statische Methoden zur Verfügung stellt um eine positive, ganze Zahl in einen Hex-String zu konvertieren, bzw. einen Hex-String in eine ganze Zahl umzuwandeln. Der statische **`char`**-Array **`digits`** enthält den gültigen Ziffernvorrat für Hexadezimalzahlen. Es sind sowohl Groß- als auch Kleinbuchstaben erlaubt. Die Algorithmen für die Konvertierung sind selber zu entwickeln.
Die beiden statischen Methoden sollen fehlerhafte Eingabezahlen erkennen und entsprechend darauf reagieren.

Erstelle die GUI-Klasse **`NumberConverterUI`** um die Methoden der BL-Klasse Klasse zu testen.



Beispiele:

```
Wählen sie aus:
  (1) Dezimal -> Hexadezimal
  (2) Hexadezimal -> Dezima
  (3) Ende
1
Dezimalzahl: 1023
Hexadezimalzahl: 3FF

Wählen sie aus:
  (1) Dezimal -> Hexadezimal
  (2) Hexadezimal -> Dezima
  (3) Ende
2
Hexadezimalzahl: 3FF
Dezimalzahl: 1023

Wählen sie aus:
  (1) Dezimal -> Hexadezimal
  (2) Hexadezimal -> Dezima
  (3) Ende
2
Hexadezimalzahl: 3XF
Ungültige Hexzahl: 3XF
```

Kompetenzen: statische Elemente, Schleifen
Umfang: 1 DP-Std