Praktikumsaufgabe 2

2.0 Lernziele

- Kontrollstrukturen, logische und arithmetische Ausdrücke anwenden
- Korrekte Verwendung von „Zählschleifen“ versus bedingungsgesteuerte Schleifen üben
- Effekte ganzzahliger Division verstehen
- Ein vorgegebenes Klassenmodell ergänzen
- Den Modulo-Operator anwenden
- Zeichenketten konvertieren
- Einen Eingabedialog für die Klasse ***Kalkulator*** erstellen

2.1 Rechnen

Implementieren Sie die Klasse ***Kalkulator*** mit den folgenden Methoden:

1. Schreiben Sie eine Methode, die die Summe der ersten n Qubikzahlen berechnet: $sum\_qubik(n) := \sum_{i=1}^{n} i^3$
2. Schreiben Sie eine Methode, die $n!$ berechnet. $n!$ heißt auch Fakultät von n: $fak(n) := \prod_{i=1}^{n} i = 1 * 2 * 3 \ldots * n$
3. Verallgemeinern Sie die Fakultätsmethode derart, dass Sie den Startwert für das Produkt beliebig wählen können. $fak(start, n) = \prod_{i=start}^{n} i = start * (start + 1) * \ldots * n$
4. Schreiben Sie eine Methode für die Berechnung des Binomialkoeffizienten. $binom(n,k) = \binom{n}{k} := \frac{n!}{k! * (n-k)!} = \frac{n*(n-1)*\ldots*(n-k+1)}{k!}$. Benutzen Sie für die Berechnung des Zählers und Nenners die Methode aus 3.
5. Schreiben Sie die Methode , die die Näherung für $\sin(x)$ berechnet: $\sin(x,n) = \sum_{i=0}^{n} (-1)^i \frac{x^{(2i+1)}}{(2*i+1)!} = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \pm \cdots \frac{x^{(2*n-1)}}{(2*n+1)!}$
6. **Zusatz optional**: Entwickeln Sie eine effizientere Lösung für 2.1.3. Berücksichtigen Sie dabei die Gleichung $\frac{x^{(2*n+1)}}{(2*n+1)!} * \frac{x^2}{(2*n+2)*(2*n+3)} = \frac{x^{(2*n+3)}}{(2*n+3)!}$, mit der sich aus dem Summanden des Schrittes n der Summand des Schrittes (n+1) berechnen lässt.
7. Schreiben Sie die Methode $n\_fuer\_fehler\_kleiner(x, eps)$, die für ein übergebenes x und Epsilon (*eps*) das *n* als Ergebnis liefert, für das gilt: $|\sin(x,n) - Math.\sin(x)| < eps$. Epsilon ist der Fehler / die maximale Abweichung vom „exakten“ Wert $Math.sin(x)$, die toleriert werden soll. Epsilon ist in der Regel sehr viel kleiner als 1, aber immer im Intervall [0,1). Hinweis: Die Aufgabe muss mir einer bedingungsgesteuerten Schleife gelöst werden.

2.2 Modulo Korrektes Berechnen und Inkrementieren von Uhrzeiten.

Sie sollen eine echte Uhr simulieren, die im Sekundentakt hochzählt. Die Uhr wird mit der Angabe von Stunden, Minuten und Sekunden erzeugt. Dabei dürfen die Stunden, Minuten und Sekunden auch jenseits des für zulässigen Wertebereiches übergeben werden. Die ***Uhr*** soll eine Methode ***tick*** implementieren, die Sekunden hochzählt und bei einem Überlauf, d.h. wenn die Anzahl der Sekunden 60 wird, die Sekunden auf 0 setzen und den Überlauf als 1 Minute darstellen. Analog soll sich die Uhr beim Überlauf von Minuten verhalten. Hier werden die Minuten auf 0 gesetzt und die Stunden um 1 erhöht. Dann muss noch sichergestellt werden, dass die Stunden bei 24 wieder bei 0 anfangen zu zählen. Es gibt verschiedene Varianten der Implementierung. Eine weitere werden wir in eine der nächsten Vorlesungen kennenlernen. Hier sollen Sie wie folgt vorgehen:

1. Die Uhr wird mit Stunden, Minuten und Sekunden erzeugt. Sie dürfen voraussetzen, dass nur ganze Zahlen übergeben werden.
    a. Im ***initialize*** werden die Werte für Stunden, Minuten und Sekunden in die Gesamtzahl von Sekunden umgerechnet und damit ein Objekt der Klasse ***Uhrzeit*** erzeugt.
    b. Das ***initialize*** soll auch das Erzeugen der Uhr mit der aktuellen Zeit ermöglichen. Arbeiten Sie dazu mit Default-Parametern für Stunden, Minuten und Sekunden und setzen Sie die Default-Werte auf ***nil***. Wenn einer der Parameter ***nil*** ist, dann sollen Stunden, Minuten und Sekunden mittels der Klasse ***Time*** ermittelt werden. ***Time.now()*** erzeugt ein Objekt mit der aktuellen Zeit. Mit den Methoden ***hour***, ***min*** und ***sec*** können Sie die benötigten Werte aus dem Time-Objekt lesen und damit wie unter a. ein Objekt der Klasse ***Uhrzeit*** erzeugen.
2. Schreiben Sie die Methode ***tick***, die die ***Uhr*** um 1 Sekunde „weiterdreht". Das macht die Uhr, indem Sie 1 Sekunde auf die ***Uhrzeit*** addiert. Implementieren Sie dazu die Methode ***+*** in der Klasse ***Uhrzeit***.
3. Schreiben Sie für die Klasse ***Uhrzeit*** die Methode ***to_s()*** , die die Sekunden in eine normalisierte Darstellung von Stunden, Minuten und Sekunden umrechnet und diese als Zeichenkette für die übliche Darstellung einer digitalen Uhrzeit aufbereitet. Die normalisierten Werte sind für Sekunden und Minuten das Intervall [0..59] und für Stunden das Intervall [0..23].
4. Damit 3. übersichtlicher wird, schreiben Sie für die Klasse ***Uhrzeit*** die Methoden ***stunden_anteil***, ***minuten_anteil***, ***sekunden_anteil***, die die Stunden-, Minuten- und Sekunden-Anteile an der Gesamt-Sekundenzahl zurückgeben. Für die Implementierung der 3 Methoden benötigen Sie den ***%*** und den ***/*** Operator.
   Beispiel: Um aus den Gesamt-Sekunden den Sekundenanteil zu berechnen, der nicht ganzzahlig in den Minutenanteil passt, berechnen Sie $GesamtSekunden\%60$. Dann bleibt für die Minuten noch $GesamtSekunden/60$ zu verteilen.

5. Schreiben Sie für die ***Uhr*** die Methode ***to_s()***, die die Uhrzeit als Zeichenkette zurückgibt.
6. Die Methode ***starten(laufzeit)*** simuliert die zeitgesteuerte ***Uhr***. Dabei wird die ***laufzeit*** in Sekunden angegeben. Diese Methode ist vorgegeben, damit Sie Ihre Uhr testen können.

Beispiel: Eine Uhr mit der aktuellen Uhrzeit, die 10 Sekunden läuft:

```
Uhr.new().starten(10)
```

```
20:42:45
20:42:46
20:42:47
20:42:48
20:42:49
20:42:50
20:42:51
20:42:52
20:42:53
20:42:54
```

Beispiel: Eine Uhr mit selbst definierter Uhrzeit, die 10 Sekunden läuft. Achten Sie auf die Nichteinhaltung der Wertebereiche bei der Erzeugung, die in der Ausgabe normalisiert korrekt dargestellt wird.

```
Uhr.new(27,74,98).starten(10)
```

```
04:15:39
04:15:40
04:15:41
04:15:42
04:15:43
04:15:44
04:15:45
04:15:46
04:15:47
04:15:48
```

Das Skript ***uhr_starter*** enthält das Erzeugen und das starten der Uhr.

2.3 Eingabedialog

Vorweg: Für diese Aufgabe sind die Klassen ***Leser***, ***Schreiber*** und ***KalkulatorDialog*** bereits vorgegeben. ***Leser*** enthält bereits eine Reihe von Methoden, die Sie verwenden sollen, ebenso ***Schreiber***. ***Leser*** ist um einige Methoden zu ergänzen. Ebenso ***KalkulatorDialog***.

Ihre Aufgabe ist es einen Eingabedialog für die Bedienung des Kalkulators zu schreiben (Klasse ***KalkulatorDialog***). Die Lese-Auswerten-Schreibschleife (Methode ***starten*** der

Klasse ***KalkulatorDialog***) wird beendet, wenn das Stop-Wort eingegeben wurde (im Bsp. „bye“).

Das Stop-Wort wird bei der Erzeugung des Kalkulator-Dialogs übergeben. Dabei sollen alle möglichen Schreibweisen des Stop-Worts erlaubt sein. Normalisieren Sie dazu sowohl das Stop-Wort als auch die Eingabe. Schreiben Sie für die Normalisierung eine Methode in der Klasse ***KalkulatorDialog***.

Der Dialog soll sich ansonsten wie folgt verhalten. In einer Lese-Auswerte-Schreibschleife soll zuerst die Aufforderung zur Eingabe ausgegeben werden:

```
Willkommen in unserem Rechenzentrum für mathematische Formeln:
Machen Sie eine der folgenden Eingaben.
1 n: um die Summe der ersten Kubikzahlen zu berechnen.
2 n: um die Fakultät zu berechnen. n ist Platzhalter für eine pos. ganze Zahl.
3 n k: um den Binomialkoeffizienten zu berechnen. n und k sind Platzhalter für eine pos. ganze Zahl.
4 x n: um die Näherung für sin(x) zu berechen. x ist Platzhalter für eine Zahl, n Platzhalter für eine pos. ganze Zahl.
5 x f: um die Anzahl der  Iterationen zu berechnen für die der Fehler der Näherung für sin(x) kleiner als ein gegebenes positives Epsilon << 1 wird.
      x ist Platzhalter für eine Zahl, f Platzhalter für eine pos Zahl << 1
Geben Sie bye ein, um den Dialog zu beenden.
```

Dieser Text ist bereits in der Instanz-Variable ***@instruktion*** hinterlegt.

Dann sollen Eingaben zeilenweise verarbeitet werden. Eine Eingabe für die Näherung von sin wäre z.B.

4 2.34 100

Besteht also aus der Auswahl der Methode (die „4“), dem 1’ten Argument (das x) und dem 2’ten Argument (das n) für die Methode ***reihe_sin(x,n)*** der Klasse ***Kalkulator***.

In der Methode ***auswerten(leser, schreiber,kalkulator)*** der Klasse ***KalkulatorDialog*** soll die Eingabe gelesen und ausgewertet werden, d.h. die zugehörige Methode aufgerufen und dazu die Argumente in den erwarteten Typ konvertiert werden. Danach soll das Ergebnis ausgegeben werden.

Zum Lesen der Eingabe hält der Dialog eine Referenz auf ein Leser-Objekt (Klasse ***Leser***). Mit der Methode ***lese()*** liest der Leser die ganze Zeile ein und merkt sich diese Zeile.

Mit der Methode ***kommando()*** der Klasse ***Leser*** bekommen Sie das erste Elemente der Zeile (im Bsp. Die „4“), mit der Methode ***argument(index)*** (index von 1..anzahl_argumente) das 2’te bis n’te Argument als Zeichenkette (im Bsp. ***argument(1)*** liefert „2.34“, ***argument(2)*** liefert „100“).

Um die Zeichenketten in Zahlen zu wandeln, müssen Sie die folgenden Methoden der Klasse ***Leser*** implementieren:

1. ***ganze_zahl?(wort)***: prüft ob eine Zeichenkette eine ganze Zahl enthält
2. ***konvertiere_in_ganze_zahl(wort)***: konvertiert eine Zeichenkette in eine ganze Zahl

3. ***konvertiere_in zahl(wort)***: konvertiert eine Zeichenkette in eine ganze Zahl, wenn die Zeichenkette eine ganze Zahl enthält sonst in eine Gleitkommazahl

Wenn die Argumente der jeweiligen Methode nicht vom erwarteten Typ sind, dann soll der ***Schreiber*** eine Fehlermeldung (Methode ***fehler_meldung_ausgeben***) ausgeben. Sonst soll die Methode aufgerufen werden und mit dem ***Schreiber*** das Ergebnis (Methode ***ergebnis_ausgeben(name_der_methode, ergebnis_der_methode,argument_1,... , argument_n)***) ausgegeben werden.

Beispiel Ausgabe für die Eingabe *4 2.34 100*:

reihe_sin(2.34,100)=0.7184647930691263

Um festzustellen welche Methode aufgerufen werden muss, verwenden Sie bitte ein **case <target>** Konstrukt mit dem Kommando als <target>. Das Ergebnis der Methode ***auswerten*** ist immer die ursprüngliche Eingabe (Methode ***eingabe*** der Klasse Leser).

Das Script ***kalkulator_dialog_starter*** erzeugt den Dialog und startet ihn mit der Methode ***starten*** der Klasse ***KalkulatorDialog***.