Hochschule Bonn-Rhein-Sieg
Fachbereich Informatik
Dipl. Inf. Sigrid Weil

# Übung 6



Thema: Iterationen

**Aufgabe 6.1**
Die *harmonische Reihe* ist definiert als unendliche Summe

$$\sum_{k=1}^{\infty} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots$$

Aus der Mathematik wissen wir, dass diese Reihe nicht kovergiert, das heißt es wird jede beliebige Zahl als Summe erreicht oder überschritten, wenn man nur genügend Summanden addiert.

Implementieren Sie eine Methode, die für einzugebendes $n$ die Teilsumme
$s = \sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots \frac{1}{n}$ berechnet.
Beispiel: für $n = 2$ sollte die Methode den Wert $s = \sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} = 1.5$ liefern

Implementieren Sie eine zweite Methode, die für eine einzugebende Grenze $s$ die Anzahl der Folgeglieder bestimmt, die addiert werden müssen, um $s$ zu erreichen oder zu übersteigen.
Beispiel: Für $s = 2$ sollte die Methode die Anzahl $n = 4$ liefern, denn
$\sum_{k=1}^{n=4} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = 2,08\overline{3} > 2$

(Achtung! Wenn Sie $s$ zu groß wählen, kann es sein, dass Ihr Programm niemals ein Ergebnis findet, weil die Summanden so klein werden, dass der Rechner sie nicht mehr von 0 unterscheidet. Außerdem könnte es passieren, dass die erforderliche Anzahl oberhalb der größten darstellbaren Integerzahl liegt. Bauen Sie also eine Sicherung in Ihre Methode ein, die auf jeden Fall spätestens nach `Integer.MAX_VALUE`-vielen Iterationen abbricht.)

**Aufgabe 6.2**
Die Fibonacci-Zahlen sind durch die **Zwei-Term-Rekursion**

$$f_0 = 1, \quad f_1 = 1, \quad f_n = f_{n-2} + f_{n-1} \text{ für } n \geq 2$$

definiert.

Schreiben Sie eine (nicht rekursive, sondern iterative) Methode, die die $n$-te Fibonacci-Zahl berechnet.

Voraussetzung für die folgende Aufgabe: Klasse `Wuerfel`. Eine Implementierung von `Wuerfel` finden Sie im Material zu dieser Übung.

**Aufgabe 6.3**
Kater Tom und Maus Jerry spielen (auf einem Spielfeld aus hintereinanderliegenden Feldern) Nachlaufen. Wie weit sie jeweils laufen, wird ausgewürfelt:
Wird eine 1, eine 2 oder eine 4 gewürfelt, läuft Jerry entsprechend viele Felder vorwärts.
Bei einer 5 muss Jerry allerdings 5 Felder zurück.
Tom springt immer, wenn eine 3 oder eine 6 gewürfelt wird, um 3 Felder nach vorne.

Tom beginnt an Position 0, Jerry erhält einen Vorsprung. Am Ende des Spielfelds liegt das rettende Mauseloch.

Das Spiel endet, wenn entweder Jerry das Mauseloch erreicht hat, oder wenn Tom Jerry eingeholt hat.

Implementieren Sie eine Klasse `Jagd`, die ein Spielfeld mit einer anzugebenden Länge erzeugt (die Länge ist also die Position des Mauselochs). Außerdem wird dem Jagd-Objekt der Vorsprung mitgegeben, den Jerry erhält. Tom beginnt immer bei Position 0.

Definieren Sie eine (private) Methode, die die aktuelle Position von Tom und Jerry auf dem Monitor anzeigt.

Eine Methode `los()` simuliert den Spielverlauf:

- Zunächst wird die (Start-)Position der beiden Spieler angezeigt.
- Dann wird (mit einem Objekt der Klasse Wuerfel) gewürfelt und die entsprechende Figur entsprechend bewegt.
- Anschließend wird wieder die aktuelle Position der beiden angezeigt.
- Bei Spielende wird der Text „gerettet!“ bzw „gefangen!“ ausgegeben.

Ein (zufälliger) Spielverlauf mit den Werten `mauseloch = 20` und `vorsprung = 5` könnte so aussehen:

Tom: 0 Jerry: 5
Tom: 0 Jerry: 9
Tom: 3 Jerry: 9
Tom: 6 Jerry: 9
Tom: 6 Jerry: 10
Tom: 6 Jerry: 12
Tom: 9 Jerry: 12
Tom: 9 Jerry: 16
Tom: 9 Jerry: 17
Tom: 9 Jerry: 21
gerettet!

Hinweis:
Auch wenn hier von „Feldern“ gesprochen wird, benötigt man für diese Aufgabe **keine** Arrays!

Voraussetzung für die folgende Aufgabe: Klasse `Wuerfel` und Eingabe via `Scanner`.
Eine Implementierung von `Wuerfel` finden Sie im Material zu dieser Übung, den Umgang mit `Scanner` können Sie in Folien99.pdf nachlesen.

**Aufgabe 6.4 (*)**
Programmieren Sie ein paar einfache Würfelspiele! Gehen Sie dabei wie folgt vor:

a) Definieren Sie eine Klasse `Spieler`: Die Klasse besitzt ein Klassenattribut vom Typ `Wuerfel`. Objekte der Spieler-Klasse verfügen über
   - einen Namen, der bei Erzeugung übergeben wird und eine get-Methode dafür
   - eine Punktzahl (zu Beginn = 0)
   - eine Methode `punkte()` zum Abfragen der Punktzahl
   - eine Methode `punktPlus()`, die diesen Wert um 1 erhöht
   - eine Methode `punktPlus(int n)`, die diesen Wert um n erhöht
   - eine Methode `reset()`, die die Punktzahl auf 0 zurücksetzt
   - eine Methode `toString()`, die den Namen des Spielers und seinen Punktestand zurückgibt
   - eine Methode `wuerfeln()`, die den Würfel wirft und den erzielten Wert zurückliefert.

b) Definieren Sie außerdem eine Klasse `Spiel`, die (zunächst) nur über eine einzige statische Methode `simplesSpiel()` verfügt. Die Methode erhält zwei Spieler-Objekte als Eingabeparameter.
   Das Spiel ist denkbar einfach: In einer Schleife sollen solange Spielrunden ausgeführt werden, wie die Abfrage
   „`Weiter? (j/n)`“ mit „j“ beantwortet wird.
   Eine Spielrunde besteht darin, dass jeder Spieler einmal wuerfelt. Wer die höhere Augenzahl geworfen hat, erhält einen Pluspunkt. Bei gleichem Wurf bekommt keiner der Spieler einen Punkt.
   In jeder Runde werden die Wurfergebnisse der beiden Spieler ausgegeben.
   Am Ende wird der Punktestand ausgegeben.

c) Schreiben Sie eine Testklasse mit einer Test-Methode, in der zwei Spieler, nämlich ein „menschlicher“ Spieler und ein „Computer-“Spieler erzeugt werden. Der menschliche Spieler soll seinen Namen via Tastatur eingeben können. Lassen Sie die beiden Spieler simplesSpiel spielen.
   Ein möglicher (und natürlich zufälliger) Spielverlauf ist in der Datei `simplesSpielAusgabe.txt` dokumentiert.

d) Implementieren Sie je nach Lust und Laune weitere Methoden in der Klasse `Spiel`, zum Beispiel:
   - Filzlaus:
     Jeder Spieler würfelt solange, bis er eine 1 gewürfelt hat, höchstens aber 8-mal. Die Anzahl der Versuche ist die Punktzahl. Ein Spieler, der innerhalb der erlaubten 8 Versuche keine 1 würfelt, hat auf jeden Fall verloren. Ansonsten gewinnt der Spieler mit der geringeren Punktzahl.
     (Es kann also Spielrunden geben, in denen beide Spieler verlieren.)
   - 6er-Pasch:
     Wie Filzlaus, mit dem Unterschied, dass nun jeder Spieler mit 2 Würfeln würfelt, solange bis er einen 6er-Pasch geworfen hat.
   - Große Hausnummer
     In jeder Runde würfelt jeder Spieler dreimal und entscheidet nach jedem Wurf, ob er die gewürfelte Zahl an der Einer-, der Zehner- oder der Hunderter-Stelle einer dreistelligen Hausnummer eintragen möchte. (Eine Position, die bereits einmal belegt wurde, kann nicht noch einmal belegt werden.) Die erreichte Hausnummer ist die in dieser Spielrunde erzielte Punktzahl.
     Für den Computer-Spieler sollen die gewürfelten Werte immer in der Reihenfolge Einer-Zehner-Hunderter eingetragen werden.
     Der Spieler mit der höheren Hausnummer gewinnt.