Heinrich Heine Universität Düsseldorf

Programmierung – WS 2025/26
Dr. Markus Brenneis
Abgabefrist für Feedback:
03.11.25, 10 Uhr

3. Übungsblatt (Lösungsvorschlag[1])

In der Woche vom 27.10.2025 finden die ersten Übungsstunden statt. In den Übungsstunden werden Präsenz-Aufgaben bearbeitet, die Ihnen bei der Bearbeitung der Übungsblätter helfen sollen.

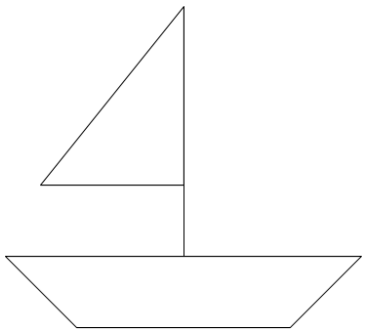
In den Tutorien dieser Woche können Sie v. a. Fragen zu Übungsblatt 2 (aber auch zu älterem und neuerem Stoff) stellen, wenn Sie welche haben – wenn Sie keine Fragen haben, verpassen Sie nichts, wenn Sie nicht zum Tutorium kommen.

Aufgabe 1: Segelboot nach VL 4

Schreiben Sie ein Programm `Ship`, welches mit den Methoden von `StdDraw` ein Segelboot (oder auch irgendwas anderes – seien Sie kreativ) zeichnet und ausgibt. Die für die Benutzung von StdDraw benötigte Datei finden Sie im Ilias.

Hinweise:

- Sie können z. B. folgendes Schiff nachmalen:



- Auch wenn Sie etwas anderes als ein Schiff zeichnen, muss Ihre Datei `Ship.java` heißen, wenn Sie sie im Abgabesystem hochladen wollen. Das erleichtert uns die Korrektur.
- Unter den schönsten[2] Abgaben verlosen wir einen kleinen Preis. Wenn Sie an der Verlosung teilnehmen wollen, müssen Sie Ihr lauffähiges Programm im Abgabesystem abgeben und um Feedback bitten.

[1] Bei den meisten Programmieraufgaben gibt es mehr als einen funktierenden Lösungsweg. Diskutieren Sie gerne untereinander Ihre Lösungsansätze und lerne Sie damit verschiedene Lösungsstrategien und verschiedene Anwendungsmöglichkeiten der Java-Funktionalitäten kennen. Ihre Abgabe müssen Sie aber final selbst formulieren/eintippen.

[2] vollkommen objektive Bewertung durch unsere Hilfskräfte

Lösungsvorschlag

- siehe `ship/Ship.java`

Aufgabe 2: Wahrheitstabelle nach VL 4

In der Rechnerarchitektur müssen Studierende oft sogenannte Wahrheitstabellen aufstellen. Wir wollen ihnen dabei helfen und ein Programm schreiben, das Wahrheitstabellen ausgeben kann.

Starten Sie mit der vorgegebenen Datei `Wahrheitstabelle.java`. Vervollständigen Sie die Vorgabe wie folgt:

- Implementieren Sie `intToBoolean` so, dass die Methode genau dann[3] `false` zurückgibt, wenn der übergebene int-Wert 0 ist.
- Die Wahrheitstabelle soll für die Funktion $f(x_1, x_2, x_3) = (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2 \wedge x_3)$ ausgegeben werden; $\vee$ steht dabei für das logische Oder, $\wedge$ für das logische Und und $\neg$ für die Negation. Passen Sie die Implementierung der Methode `f` an, sodass sie der hier angegebenen Definition f entspricht.

Aufgabe 3: Cosinus nach VL 4

Der Prozessor eines Computers kann typischerweise nicht direkt die Cosinus-Funktion berechnen. Eine Möglichkeit, um trotzdem Cosinus-Werte berechnen zu können, ist das Zurückführen auf elementare Rechenoperationen. Die Cosinusfunktion kann z. B. mithilfe der sogenannten Taylor-Entwicklung näherungsweise berechnet werden (x im Bogenmaß):

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \cdots$$

Für diese Aufgabe sollen Sie nur die ersten 11 Terme (also bis einschließlich $\frac{x^{20}}{20!}$) der Nährungsformel benutzen.[4]

Diese Näherung wird außerhalb des Intervalls $[0, 2\pi]$ sehr schnell ungenau, sodass man x auf dieses Intervall abbilden sollte. Dazu können die Achsensymmetrie und die Periodizität der Cosinusfunktion ausgenutzt werden.

Schreiben Sie ein Programm `Cosinus` mit einer Methode `static double cosinus(double x)`, welche $\cos(x)$ mithilfe der oben angegebenen Näherung berechnet und den berechneten Wert zurückgibt.

Für diese Aufgabe dürfen Sie aus `Math` ausschließlich die Konstante `Math.PI` benutzen. Weitere eingebaute Math-Funktionalitäten (insbes. `Math.cos` und `Math.pow`) oder -Konstanten dürfen in Ihrem Programm nicht verwendet werden.

[3] „Genau dann“ bedeutet, dass in allen anderen Fällen *nicht* `false` zurückgegeben werden soll, sondern `true`.

[4] Wenn Sie das Summenzeichen aus der Mathematik schon kennen: Als Summenformel lässt sich die zu programmierende Formel folgendermaßen ausdrücken:

$$\cos(x) \approx \sum_{n=0}^{10} \frac{(-1)^n}{(2n)!} \cdot x^{2n}$$

Tipps:

- Vollziehen Sie im ersten Schritt nach, was die Aufgabe von Ihnen verlangt: Was ist Eingabe? Was ist Ausgabe/Ergebnis? Wie funktioniert die Berechnung, insbesondere außerhalb von $[0, 2\pi]$? Wenn Sie nicht mehr wissen, was Achsensymmetrie und Periodizität bedeuten, recherchieren Sie die Begriffe.
- Versuchen Sie, Teilprobleme zu identifizieren, die Sie jeweils in eigenen Methoden lösen. Schreiben Sie z. B. eine eigene Methode für die Fakultätsberechnung, um diese dann später in der `cosinus`-Methode zu verwenden.
- Testen Sie jeden neuen Codeteil mit unterschiedlichen Eingaben: Funktioniert die Fakultätsberechnung? Funktioniert die Potenzberechnung? Ist das Gesamtergebnis am Ende richtig? Schreiben Sie zum Testen eine `main`-Methode.
- Evtl. hilft Ihnen dieser Selbsttest auf der Kursseite zum Entwickeln und Testen kleiner Methoden weiter.
- Sie sollen den Cosinus nicht exakt berechnen, sondern die Nährung mithilfe der ersten 11 Terme im Intervall $[0, 2\pi]$ benutzen.
- Außerhalb einer Übungsaufgabe würden Sie direkt `Math.cos` aufrufen. Wir wollen mit dieser Aufgabe das Umsetzen von mathematischen Formeln in Java-Code üben.[5]

Aufgabe 4: Pascalsches Dreieck nach VL 5

Das Pascalsche Dreieck[6] ist eine dreieckige, systematische Anordnung von Zahlen. Die Zeilen des Dreiecks beginnen und enden außen mit einer 1. Die übrigen Zahlen ergeben sich als Summe der beiden darüberliegenden Zahlen:

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

Eine Anwendung des Dreiecks ist der binomische Lehrsatz zum Auflösen von Potenzen der Form $(x + y)^n$: Die Faktoren ergeben sich durch die n-te Zeile im Dreieck:

$$(x + y)^2 = \mathbf{1}x^2 + \mathbf{2}xy + \mathbf{1}y^2$$
$$(x + y)^3 = \mathbf{1}x^3 + \mathbf{3}x^2y + \mathbf{3}xy^2 + \mathbf{1}y^3$$

Wir wollen nun ein Programm `PascalTriangle` schreiben, um das Pascalsche Dreieck zu berechnen und auszugeben. Schreiben Sie dazu zunächst eine **rekursive** Methode

[5] Die Umsetzung von `Math.cos` im JDK selbst (unter Verwendung der Programmiersprache C) benutzt übrigens ein ähnliches, aber weiter optimiertes Verfahren: https://github.com/openjdk/jdk/blob/05a764f4ffb8030d6b768f2d362c388e5aabd92d/src/java.base/share/native/libfdlibm/k_cos.c#L32

[6] https://de.wikipedia.org/wiki/Pascalsches_Dreieck

`static int pascalRecursive(int zeile, int spalte)`[7], die den Wert an der entsprechenden Zeilen- und Spaltenposition mithilfe der folgenden Formel berechnet:

$$p(z,s) = \begin{cases} 1 & \text{falls Randfeld, d. h. } s = 0 \text{ oder } s = z \\ p(z-1, s-1) + p(z-1, s) & \text{sonst} \end{cases}$$

$p(z,s)$ bestimmt den Wert an Zeilenposition z und Spaltenposition s (jeweils gezählt ab 0).

Schreiben Sie dann eine Methode `static void pascalTriangle(int n)`, die ein Pascalsches Dreieck bis einschließlich Zeile n auf der Standardausgabe ausgibt; die erste Zeile zählt als Zeile 0. Die einzelnen Werte sollen in der Ausgabe mit jeweils genau einem Leerzeichen getrennt werden. Beispiel für $n = 5$:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

Testen Sie die Korrektheit beider Methoden für verschiedene Werte, indem Sie die Methoden in der `main`-Methode aufrufen.

[7] Die Namen der Parameter dürfen Sie frei wählen, die restliche Vorgabe ist für die Abgabe einzuhalten. Diese Regel gilt in Java immer, wenn Sie für andere Personen Methoden implementieren sollen.