Programmieren II (Java)

1. Praktikum: Grundlagen

Sommersemester 2024
Christopher Auer

Abgabetermine

Lernziele

- Erstes Beschnuppern von Java
- Arbeiten mit Kontrollstrukturen und primitiven Datentypen
- Arithmetik
- Implementieren einer Konsolenanwendung
- Implementierung eines Algorithmus nach einer Spezifikation

Hinweise

- Sie dürfen die Aufgaben *alleine* oder zu *zweit* bearbeiten und abgeben
- Sie müssen *4 der 5* Praktika bestehen
- *Kommentieren* Sie Ihren Code
  - Jede *Methode* (wenn nicht vorgegeben)
  - *Wichtige* Anweisungen/Code-Blöcke
  - *Nicht kommentierter* Code führt zu *Nichtbestehen*
- Bestehen Sie eine Abgabe *nicht* haben Sie einen *zweiten Versuch*, in dem Sie Ihre Abgabe *verbessern müssen*.
- *Wichtig*: Sie sind einer *Praktikumsgruppe* zugewiesen, *nur* in dieser werden Ihre Abgaben *akzeptiert*!

Aufgabe 1: Java zu Fuß

Erstellen Sie eine Java-Datei mit folgendem Inhalt und dem Namen `HelloJava.java`.

```java
public class HelloJava{
  public static void main(String[] args){
    System.out.println("Hello Java!");
  }
}
```
HelloJava.java

- ☑ Installieren Sie das JDK („Java Development Kit“) von Oracle oder OpenJDK
- ☑ Starten Sie eine Kommandozeile und navigieren Sie (mit `cd`) in das Verzeichnis, in dem die Datei `HelloJava.java` liegt.
- ☑ Übersetzen Sie die Datei folgenden Kommando in eine `.class`-Datei:

```
javac HelloJava.java
```

- ☑ Führen Sie das Programm aus mit

```
java HelloJava
```

  Das Programm sollte `Hello Java!` ausgeben.

Hinweise

- Sie benötigen für diese Aufgabe keine Entwicklungsumgebung.
- Diese Aufgabe müssen Sie *nicht abgeben*

Aufgabe 2: Volumenberechner

Schreiben Sie ein Java-Programm `Flaechenberechner.java`, das das Volumen verschiedener geometrischer Objekte berechnen kann. Das Programm wird dabei über Kommandozeilenparameter gesteuert:

- Wird *kein* Kommandozeilenparameter übergeben (`args.length == 0`), soll folgende Ausgabe geschehen:

```
Aufruf: java Flaechenberechner
Verfügbare Berechnungen:
Kugel: Radius
Pyramide: Grundseite Höhe
Quader: Länge Breite Höhe
```

- Ansonsten ergibt die *Anzahl* der Parameter das zu berechnende Volumen an:

Objekt	`args.length`	Dimensionen	Volumen
Kreis	1	Radius r	$\frac{4}{3}\pi r^3$
Pyramide	2	Grundseite g, Höhe h	$\frac{1}{3} \cdot g^2 \cdot h$
Quader	3	Länge l, Breite b, Höhe h	$l \cdot b \cdot h$

- Geben Sie das berechnete Volumen auf der Konsole aus, z.B.:

```
gradlew Volumenberechner --args="1 2"
Pyramidenvolumen: 0.666667
```

Oder:

```
gradlew Volumenberechner --args="4.11"
Kugelvolumen: 290.813173
```

Bonusaufgabe In dem bisherigen Programm werden fehlerhafte Eingaben noch nicht abgefangen. Was passiert z.B. wenn die Anzahl der Parameter größer als vier ist oder eine Eingabe keiner Zahl entspricht?

- ☑ Probieren Sie verschiedene fehlerhafte Eingaben aus und vollziehen Sie nach, zu welchem Fehler es kommt.
- ☑ Erweitern Sie Ihr Programm so, dass es möglichst viele dieser fehlerhaften Eingaben abprüft und mit einer Fehlermeldung quittiert.

Aufgabe 3: Einmaleins

Wir entwickeln ein kleines Konsolenprogramm, mit dem man das kleine Einmaleins üben kann. Dabei soll die Ein- und Ausgabe wie folgt aussehen (Eingaben sind rot):

```
Wieviele Aufgaben wohlen Sie rechnen? (keine negativen Zahlen)
5
Was ist 5 * 8?
40
Richtig!
Was ist 6 * 10?
60
Richtig!
Was ist 6 * 7?
44
Leider nicht richtig!
Was ist 3 * 1?
3
Richtig!
Was ist 2 * 9?
18
Richtig!
4 von 5 Aufgaben korrekt (80%)
Gut gemacht!
```

Das heißt, zuerst wird eingelesen *wieviele* Aufgaben gerechnet werden sollen, dann werden *zufällige Aufgaben* generiert und schließlich wird eine *Auswertung* ausgegeben. Dabei soll bei der Auswertung am Schluss ein Hinweis ausgegeben werden, der vom erreichten Anteil richtiger Lösungen abhängt für Werte über 99%, 90%, 75%, 50% und 25%. Implementieren Sie das Programm in der `main`-Methode einer Klasse `Einmaleins`!

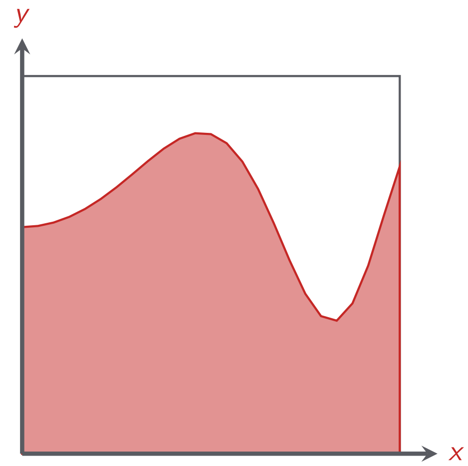
Hinweise, wenn Sie nicht wissen, wie Sie anfangen sollen:

- Gehen Sie in *kleinen Schritten* vo und testen Sie zwischen den Schritten, bevor Sie fortfahren:
  - Einlesen der Anzahl der Aufgaben
  - zufälliges Generieren *einer Aufgabe*
  - Einlesen der Antwort und Abgleich mit richtigen Ergebnis für *eine Aufgabe*
  - Abfragen *aller Aufgaben*
  - Ausgeben der *Auswertung*
- Verwenden Sie zum Einlesen der Benutzereingaben die Klasse `Scanner`: Erstellen Sie eine `Scanner`-Instanz über:

  ```
  Scanner scanner = new Scanner(System.in);
  ```
- Über die Methode `Scanner.nextInt()` können Sie eine Zahl vom Nutzer einlesen.
- Die Methode `Math.random()` erzeugt eine **`double`**-Zufallszahl zwischen 0 und 1 (ausgeschlossen). Überlegen Sie sich, wie Sie daraus eine **`int`**-Zufallszahl zwischen 0 und 9 (eingeschlossen) erzeugen können, um eine zufällige Einmaleins-Aufgabe zu erzeugen.
- Was passiert wenn der Nutzer eine *fehlerhafte* Eingabe macht? Welche fehlerhaften Eingaben gibt es und wie können Sie diese *abfangen*?

Aufgabe 4: Nummerische Annäherung von Integralen

In dieser Aufgabe nähern wir das Integral einer reellen Funktion an. Als Vereinfachung betrachten wir die Funktion im Intervall $[0, 1]$ und verlangen, dass die Funktionswerte in dem Intervall ebenfalls im Intervall $[0, 1]$ liegen.



Die rot schattierte Fläche entspricht dem Integral der Funktion im Intervall $[0, 1]$.

Es ist im Allgemeinen nicht so einfach das Integral einer Funktion exakt zu berechnen, aber glücklicherweise gibt es verschiedene Näherungsverfahren. Wir betrachten hier die *Monte-Carlo-Integration*: Bezeichnen wir mit A_f den Flächeninhalt unterhalb der Funktion begrenzt durch die x-Achse und mit A_Q den Flächeninhalt des Quadrats $[0, 1] \times [0, 1]$. Dann ist A_f der Wert des Integrals und $A_Q = 1 \cdot 1 = 1$. Erzeugt man zufällig[1] N Punkte innerhalb des Quadrats und zählt wie oft Punkte *unterhalb* der Funktion landen (bezeichnet durch N_f, schattierter Bereich), dann gilt für *große N*:

$$\frac{N_f}{N} \approx \frac{A_f}{A_Q}$$

Wenn wir für $A_q = 1$ einsetzen, bekommen wir:

$$\frac{N_K}{N} \approx A_f$$

Den Wert des Integrals kann man also annähern indem man zählt wieviele Punkte unterhalb der Kurve liegen und dies durch die Anzahl aller Punkte teilt.

- ☑ Erstellen Sie eine Java-Klasse `MonteCarloIntegration` mit einer `main`-Methode.
- ☑ Implementieren Sie eine *statische Methode* `public static double function(double x)`, die den Funktionswert berechnet. Als sehr einfaches Beispiel, verwenden Sie die Funktion:

  $$f(x) = x$$

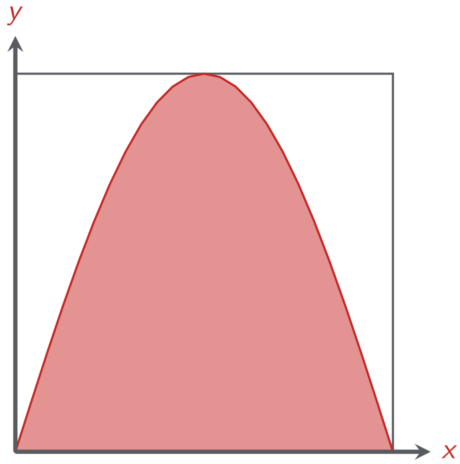
  Dies entspricht einer Gerade mit *Steigung* 1, die im Ursprung beginnt. Das Integral hat hier den Wert 0.5.

[1] Für Stochastik-Spezialisten: *uniform verteilt* und *unabhängig*

Als Vorschlag für eine interessantere Funktion, verwenden Sie:

$$f(x) = \sin \pi \cdot x$$

Die Kurve sieht wie folgt aus:



Und der (exakte) Wert des Integrals hat den Wert $\frac{2}{\pi} \approx 0.6366$.[2]

- ☑ Um die Anzahl der Iterationen zu begrenzen, deklarieren Sie eine Konstante `MAX_ITERATIONS` in der Klasse `MonteCarloIntegration` mit dem Wert `100_000`. Deklarieren Sie zusätzliche eine Konstante `MIN_CHANGE` mit dem Wert 10^{-5}. Die Funktion der Variable wird weiter unten klar.
- ☑ Implementieren Sie folgenden Algorithmus zur Annäherung des Integrals in der `main`-Methode
  - ▶ Deklarieren Sie drei lokale Variablen mit geeigneten Datentypen und Initialwerten:
    - ▶ `allPoints` — Anzahl der Punkte (entspricht der Anzahl der bisherigen Iterationen)
    - ▶ `pointsUnderCurve` — Anzahl der Punkte *unter dem Funktionsgraphen*
    - ▶ `double approxInt` — *Annäherung des Integrals*
  - ▶ Solange die Anzahl der Iterationen *kleiner* als `MAX_ITERATIONS` ist *und* der Absolutbetrag der Änderung des Werts von `approxInt` im Vergleich zur vorherigen Iteration *mindestens* `MIN_CHANGE`, wiederhole:
    - ▶ Erzeuge eine zufällige x-Koordinate zwischen 0 und 1 (verwenden Sie dazu `Math.random()`)
    - ▶ Erzeuge eine zufällige y-Koordinate zwischen 0 und 1
    - ▶ Erhöhen Sie den Wert von `allPoints` um `1`
    - ▶ Wenn der y-Wert *kleiner oder gleich* dem Funktionswert an der Stelle x ist, so erhöhen Sie `pointsUnderCurve` um `1`.
    - ▶ Berechnen Sie anhand der obigen Formel `approxInt`.
    - ▶ Geben Sie das aktuelle Ergebnis aus, z.B.:

      ```
      Iteration 36257: 0.63745 (0.000010)
      ```

      Die bisherigen Annäherung des Integrals soll mit *fünf Nachkommastellen* dargestellt werden.
- ☑ Testen Sie Ihr Programm *mehrmals*! Manchmal kommt es zu einem seltsamen Verhalten:

  ```
  Iteration 1: 1.00000 (1.000000)
  Iteration 2: 1.00000 (0.000000)
  ```

  Warum tritt das Verhalten auf und wie könnten Sie es *beheben* bzw. zumindest *unwahrscheinlicher* machen?

[2] D.h. wenn Sie den Kehrwert des Integrals mit 2 multiplizieren, sollte ungefähr π herauskommen.