

# Programmieren II (Java)

## 1. Praktikum: Grundlagen

Sommersemester 2022

Christopher Auer, Tobias Lehner



### Abgabetermine

### Lernziele

- ▶ Erstes Beschnuppen 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*
  - ▶ Beachten Sie dazu die Hinweise auf der [Moodle-Kursseite](#)

## Aufgabe 1: Java zu Fuß ★★

Erstellen Sie eine Java-Datei mit folgendem Inhalt und dem Namen `HelloJava.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: *R*-Faktor ★★ bis ★★

In dieser Aufgabe beschäftigen wir uns mit dem  $\rightarrow$  *R*-Faktor (auch „*Basic Reproduction Number*“). Der Faktor gibt an, wieviele weitere Personen eine mit einem Virus (z.B. COVID-19) angesteckte Person im Mittel ansteckt. Dabei gilt: Sind  $N_t$  Personen zum Zeitpunkt  $t$  infiziert, so sind zum Zeitpunkt  $t + 1$  im Schnitt  $N_{t+1} = R \cdot N_t$  infiziert. Gilt  $R > 1$  so haben wir *exponentielles Wachstum*. Für  $R < 1$  sinkt die Anzahl der Infizierten und für  $R \approx 1$  bleibt die Anzahl der Infizierten ungefähr gleich.

Natürlich handelt es sich hierbei um ein *sehr einfaches epidemiologisches Modell*!

### Simulation ★★

- ✓ Erstellen Sie eine Java-Klasse `RFactor` mit einer `main`-Methode.
- ✓ Deklarieren Sie drei Variablen in `main`-Methode
  - ▶ `infected` — Anzahl der infizierten Personen (**double**, initialer Wert 100)
  - ▶ `rFactor` — *R*-Faktor (**double**, Wert 1.1)
  - ▶ `iterations` — Anzahl der auszuführenden Iterationen (**int**, Wert 10)
- ✓ Ergänzen Sie Ihr Programm um die Berechnung der Anzahl infizierter Personen für `iterations` viele Durchgänge. Die Ausgabe soll dabei wie folgt aussehen:

```
Initial: 100.000000
Iteration 1: 110.000000
Iteration 2: 121.000000
Iteration 3: 133.100000
Iteration 4: 146.410000
Iteration 5: 161.051000
Iteration 6: 177.156100
Iteration 7: 194.871710
Iteration 8: 214.358881
Iteration 9: 235.794769
Iteration 10: 259.374246
```

### Kommandozeilenparameter ★★

Die drei Parameter `infected`, `rFactor` und `iterations` sollen über die Kommandozeile als Parameter in dieser Reihenfolge an das Hauptprogramm übergeben werden:

```
java RFactor 10 0.9 5
Initial: 10.000000
Iteration 1: 9.000000
Iteration 2: 8.100000
Iteration 3: 7.290000
Iteration 4: 6.561000
Iteration 5: 5.904900
```

Diese Kommandozeilenparameter sind im *Array* `String[] args` zu finden. Der  $\rightarrow$  `String arg[0]` beinhaltet dabei den Parameter `infected`, usw.

- ✓ Die Kommandozeilenparameter sind vom Typ `String` und müssen zunächst in die Datentypen von `infected`, `rFactor` und `iterations` **umgewandelt** („**geparsed**“) werden. Dazu gibt es die Methoden `Double.parseDouble(String s)` und `Integer.parseInt(String s)`. Passen Sie Ihre `main`-Methode so an, dass die drei Variablen mit den Werten von der Kommandozeile initialisiert werden.
- ✓ Erweitern Sie Ihr Programm wie folgt: Sollte  $R < 1$  sein und die Anzahl der infizierten Personen unter 0.1 fallen, brechen Sie die Iteration mit einer Meldung ab:

```
java RFactor 5 0.8 100
Initial: 5.000000
Iteration 1: 4.000000
Iteration 2: 3.200000
...
Iteration 17: 0.112590
Iteration 18: 0.090072
The pandemic is over!
```

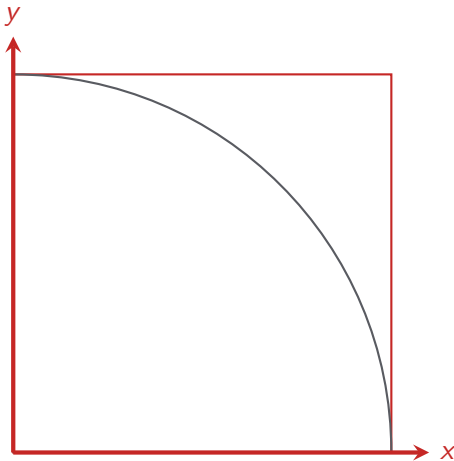
### Fehlerbehandlung ☆☆ bis ☆☆

- ✓ Sollte der Nutzer die **falsche Anzahl** an Parametern übergeben, geben Sie eine **Fehlermeldung** aus und beenden Sie das Programm. Hinweis: Die Anzahl der Parameter ist durch `args.length` gegeben!
- ✓ Die einzige fehlerhafte Eingabe, die wir bisher prüfen ist die **Anzahl der Parameter**. Überlegen Sie sich, welche Arten von fehlerhaften Eingaben es noch geben könnte! Führen Sie Ihr Programm damit aus und beobachten Sie, wie es sich verhält.
- ✓ Implementieren Sie **zumindest eine weitere** Behandlung eines ungültigen Eingabeparameters.



### Aufgabe 3: Numerische Annäherung von $\pi$ ★★

In dieser Aufgabe nähern wir die Kreiszahl  $\pi$  über ein sogenanntes ☐ Monte Carlo Verfahren an. Dazu brauchen wir ein paar Vorbereitungen: Einem Quadrat der Seitenlänge 1 ist ein Viertelkreis einbeschrieben:



Die Fläche des Quadrats ist  $A_Q = 1 \cdot 1 = 1$ , die Fläche des Viertelkreises ist  $A_K = \frac{1}{4} \pi \cdot 1^2 = \frac{\pi}{4}$ .

Erzeugt man nun zufällig<sup>1</sup>  $N_Q$  Punkte innerhalb des Quadrats und zählt wie oft Punkte *innerhalb* des Viertelkreises liegt ( $N_K$ ), dann gilt für *große*  $N$ :

$$\frac{N_K}{N_Q} \approx \frac{A_K}{A_Q}$$

Die rechte Seite können wir einsetzen und bekommen:

$$\frac{N_K}{N_Q} \approx \frac{\frac{\pi}{4}}{1} = \frac{\pi}{4}$$

Und damit gilt:

$$\pi \approx 4 \cdot \frac{N_K}{N_Q}$$

Wir können also die Kreiszahl  $\pi$  annähern, indem wir zählen wie oft Punkte innerhalb des Kreises liegen.

- ✓ Erstellen Sie eine Java-Klasse `MonteCarloPi` mit einer `main`-Methode.
- ✓ Implementieren Sie folgenden Algorithmus zur Annäherung von  $\pi$  in der `main`-Methode
  - ▶ Deklarieren Sie drei Variablen mit geeigneten Datentypen und Initialwerten:
    - ▶ `inSquare` — Anzahl der Punkte *im Quadrat* (entspricht der Anzahl der Iterationen)
    - ▶ `inCircle` — Anzahl der Punkte *im Kreis*
    - ▶ `double approxPi` — *Annäherung von  $\pi$*
  - ▶ *Solange* der Absolutbetrag (☐ `Math.abs`) der Differenz von `approxPi` und ☐ `Math.PI` größer als  $10^{-5}$  ist, wiederhole...
    - ▶ Erzeuge eine zufällige  $x$ -Koordinate zwischen 0 und 1 (verwenden Sie dazu ☐ `Math.random()`)
    - ▶ Erzeuge eine zufällige  $y$ -Koordinate zwischen 0 und 1

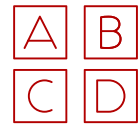
<sup>1</sup>Für Stochastik-Spezialisten: *uniform verteilt* und *unabhängig*

- ▶ *Erhöhe* inSquare um 1
- ▶ Gilt  $\sqrt{x^2 + y^2} \leq 1$ , so liegt der Punkt *innerhalb* des Kreises. In diesem Fall erhöhen Sie inCircle um 1. Hinweis: `Math.sqrt` berechnet die Wurzel.<sup>2</sup>
- ▶ Berechnen Sie approxPi mit obiger Formel.
- ▶ Geben Sie das aktuelle Ergebnis aus, z.B.:


Iteration 30263: 3.138222

---

<sup>2</sup>Für Performance-Spezialisten: Man kann auch *ohne* Wurzelberechnen herausfinden, der Punkt im Kreis liegt.




## Aufgabe 4: Wordle ★★ bis ★★

Ende 2021 und Anfang 2022 war das Spiel  Wordle von Josh Wardle im Web sehr beliebt. In dem Spiel muss ein *Lösungswort* (z.B. *TASSE*) aus *fünf Buchstaben* in *sechs Versuchen* geraten werden. Dazu gibt man ein Wort (z.B. *SAUER*) ein und das Spiel zeigt:

- ▶ Buchstaben im Lösungswort *an der richtigen Stelle* (z.B. *A*)
- ▶ Buchstaben im Lösungswort *an der falschen Stelle* (z.B. *S*, *E*)
- ▶ Buchstaben *nicht im Lösungswort* (z.B. *U*, *R*)

In dieser Aufgaben implementieren wir eine Konsolenversion dieses Spiels:

- ✓ Erstellen Sie eine Java-Klasse mit dem Namen `Wordle` und einer `main`-Methode.
- ✓ Deklarieren Sie in einem String das *Lösungswort* als *lokale Variable*.
- ✓ Lesen Sie in Ihrem Programm die Versuche des Nutzers mit Hilfe der  `Scanner`-Klasse ein. Verwenden Sie dazu die Methode `nextLine`:

```
Guess the secret word! You have 6 guesses!  
SAUER<ENTER> Benutzereingabe
```

- ✓ Zeigen Sie dann für jeden eingegebenen Buchstaben die oben angegebenen Hinweise an (*Lösungswort TASSE*). Ignorieren Sie dabei Groß-/Kleinschreibung. Beispiel:

```
sauer<ENTER>  
?A!?!?
```

- ▶ Zeigen Sie den eingegebenen Buchstaben an, wenn er an der richtigen Stelle steht.
- ▶ ? für einen Buchstaben *im Lösungswort* an der *falschen Stelle*.
- ▶ ! für einen Buchstaben *nicht Lösungswort*.
- ✓ Sollte der Nutzer *nicht exakt 5 Buchstaben eingeben*, zeigen Sie eine Fehlermeldung (und zählen Sie die Eingabe nicht als einen Versuch):

```
pass<ENTER>  
Please enter exactly 5 letters
```

- ✓ Zeigen Sie dem Nutzer an, wenn er das Lösungswort *geraten hat*:

```
TASSE  
You guessed right!
```

- ▶ Zeigen Sie dem Nutzer nach *sechs Versuchen* an, dass er keine Versuche mehr hat:

```
...  
sauer<ENTER>  
?A!?!?  
No tries left!
```

- ▶ Hilfreiche Methoden:
  - ▶ `String.length()` liefert die Länge des Strings.
  - ▶ `String.charAt(int index)` liefert den `char` an der Stelle `index` ( $0 \leq \text{index} < \text{Länge des Strings}$ ).
  - ▶ `System.out.print(...)` macht eine Ausgabe *ohne Zeilenvorschub*, `println` *mit Zeilenvorschub*.
  - ▶ `Character.toUpperCase(char c)` wandelt einen `char` in einen Großbuchstaben um.