



Praktikum zu
**Einführung in die Informatik für
LogWings, WiMas und MedPhys**
Wintersemester 2020/21

Übungsblatt 7
Besprechung:
11.–15.1.2021 (KW 2)

Vorbereitende Aufgaben

Aufgabe 7.1: Strings – Vorbereitung

Der Datentyp **String** sieht wie ein primitiver Datentyp aus, ist er aber in Wirklichkeit nicht. Dies kann man am Beispiel des Vergleichens zweier Strings verdeutlichen.

- a) Warum ist es normalerweise nicht sinnvoll, Strings mit dem `==`-Operator zu vergleichen?

== vergleicht nur die Speicheradressen und nicht den Inhalt

- b) Wie vergleicht man stattdessen den Inhalt zweier Strings?

mit der equals-Methode. Beispiel: eingabe.equals(password)

Aufgabe 7.2: Fakultät rekursiv – Vorbereitung

Auf Aufgabenblatt 5 haben wir die Fakultät einer Zahl iterativ berechnet. Die Fakultätsfunktion eignet sich auch sehr gut für rekursive Funktionen. Überlegen Sie sich eine rekursive Definition zur Berechnung der Fakultät.

$$\text{factorial}(n) = \begin{cases} 1 & \text{falls } n = 0 \\ n \cdot \text{factorial}(n - 1) & \text{sonst} \end{cases}$$

Präsenzaufgaben

Aufgabe 7.3: „ganz einfache“ Rekursion

In dieser Aufgabe wollen wir uns mit dem Verhalten von rekursiven Programmen beschäftigen. Erstellen Sie eine Klasse **EasyRecursion** dafür.

- a) Schreiben Sie eine rekursive Funktion **descendingPrint**, die die Zahlen von 1 bis **n** in **absteigender** Reihenfolge (von **n** bis 1) ausgibt. **n** ist der Parameter der Funktion.
- b) Kopieren und verändern Sie die Funktion so, dass die Zahlen **aufsteigend** (von 1 bis **n**) ausgegeben werden. Nennen Sie die neue Funktion **ascendingPrint**.

Komplettlösung

```
public static void descendingPrint(int n) {
    System.out.println(n);
    if (n > 1) {
        descendingPrint(n - 1);
    }
}

public static void ascendingPrint(int n) {
    if (n > 1) {
        ascendingPrint(n - 1);
    }
    System.out.println(n);
}
```

Aufgabe 7.4: Fakultät rekursiv

- a) Beschreiben Sie schrittweise von Hand, wie die Fakultät von 5 entsprechend Aufgabe 7.2 berechnet wird.

Lösung

$$\begin{aligned} & \text{factorial}(5) \\ &= 5 \cdot \text{factorial}(4) \\ &= 5 \cdot (4 \cdot \text{factorial}(3)) \\ &= 5 \cdot (4 \cdot (3 \cdot \text{factorial}(2))) \\ &= 5 \cdot (4 \cdot (3 \cdot (2 \cdot \text{factorial}(1)))) \\ &= 5 \cdot (4 \cdot (3 \cdot (2 \cdot (1 \cdot \underbrace{\text{factorial}(0)}_{=1})))) \\ &= 5 \cdot (4 \cdot (3 \cdot (2 \cdot (1 \cdot 1)))) \\ &= 5 \cdot (4 \cdot (3 \cdot (2 \cdot 1))) \\ &= 5 \cdot (4 \cdot (3 \cdot 2)) \\ &= 5 \cdot (4 \cdot 6) \\ &= 5 \cdot 24 \end{aligned}$$

= 120

- b) Nun sollen Sie Ihre Überlegungen aus Aufgabe 7.2 implementieren. Schreiben Sie dazu eine Funktion **factorial** in der Klasse **RecursiveFactorial**. Sie soll die Fakultät einer Zahl **rekursiv** berechnen. Rufen Sie sie in der **main**-Funktion auf und geben Sie das Ergebnis aus.

Hinweis: Hier sollen Sie keine Ausgabe in den rekursiven Funktionen verwenden.

Lösung

```
public static int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Aufgabe 7.5: Potenzen

Nun wollen wir eine rekursive mathematische Definition implementieren. Um die Potenz b^e auszurechnen, kann man folgende Funktion benutzen ($b \in \mathbb{R}$ und $e \in \mathbb{N}$, mit b als Basis und e als Exponent):

$$\text{pow}(b, e) = \begin{cases} 1 & \text{falls } e = 0 \\ (\text{pow}(b, e/2))^2 & \text{falls } e \text{ gerade} \\ b \cdot \text{pow}(b, e - 1) & \text{sonst} \end{cases}$$

Machen Sie sich am Beispiel 3^5 klar, dass diese Funktion tatsächlich funktioniert.

$$\underline{3^5 = 3 \cdot 3^4 = 3 \cdot (3^2)^2 = 3 \cdot 3^2 \cdot 3^2 = 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \text{ (die ganzen } \cdot 1 \text{ weggelassen)}}$$

Implementieren Sie diese Funktion in der Klasse **Pow**. Ignorieren Sie bei Ihrer Implementierung den Fall, dass der Exponent auch negativ sein kann.

Lösung

```
1 public class Pow {
2     public static double pow(double b, int e) {
3         if (e == 0) {
4             return 1;
5         } else if (e % 2 == 0) {
6             double tmp = pow(b, e / 2);
7             return tmp * tmp;
8         } else {
9             return b * pow(b, e - 1);
10        }
11    }
12 }
```

```

13     public static void main(String[] args) {
14         System.out.println(pow(2.0, 10));
15     }
16 }

```

Aufgabe 7.6: Basisumrechnung

In den Aufgaben von Blatt 1 haben Sie bereits gelernt, wie man Dezimalzahlen in das Binärsystem umrechnet. Implementieren Sie in der Klasse **EasyRecursion** dafür folgende Funktion:

```
public static void printInBinary(int n) {...}
```

Die Funktion soll eine Dezimalzahl n binär ausgeben. Nehmen Sie an, dass n nicht negativ ist.

Lösung

```

public static void printInBinary(int n) {
    int rest = n % 2;
    int quotient = n / 2;
    if(quotient > 0) {
        printInBinary(quotient);
    }
    System.out.print(rest);
}

```

Ergänzende Aufgaben

Aufgabe 7.7: Euklidischer Algorithmus

Abschließend wollen ein rekursives Programm analysieren und optimieren. Der *größte gemeinsame Teiler* (ggT) zweier natürlicher Zahlen m und n ist die größte Zahl, durch die sowohl m als auch n teilbar ist. Ein Algorithmus zur Berechnung des ggT ist der *euklidische Algorithmus*:

$$\text{ggT}(m, n) = \begin{cases} n & \text{falls } m = 0 \\ m & \text{falls } n = 0 \\ \text{ggT}(m - n, n) & \text{falls } m > n \\ \text{ggT}(m, n - m) & \text{sonst} \end{cases}$$

In diesem wird immer abwechselnd die kleinere Zahl von der größeren abgezogen, bis eine von beiden 0 ergibt. Die andere Zahl ist dann der ggT. Eine rekursive Implementierung könnte so aussehen:

```

public static int euclid(int m, int n) {
    if (m == 0) {
        return n;
    } else if (n == 0) {
        return m;
    } else if (m > n) {
        return euclid(m - n, n);
    } else {

```

```
    return euclid(m, n - m);  
  }  
}
```

Diese Implementierung funktioniert, ist aber ineffizient. Im Laufe der Veranstaltung haben Sie eine Möglichkeit kennengelernt, die Rechenschritte dieses Algorithmus zu verkürzen.

Berechnen Sie per Hand `euclid(15, 42)`.

$$\begin{aligned} \underline{\text{euclid}(15, 42) = \text{euclid}(15, 27) = \text{euclid}(15, 12) = \text{euclid}(3, 12)} \\ \underline{= \text{euclid}(3, 9) = \text{euclid}(3, 6) = \text{euclid}(3, 3) = \text{euclid}(3, 0) = 3} \end{aligned}$$

Was fällt Ihnen auf? Angenommen m sei nach der letzten Vertauschung größer als n , was haben Sie berechnet, wenn in diesem Rekursionsschritt m nicht mehr größer als n ist?

$$\underline{m \bmod n}$$

Geben Sie eine rekursive Implementierung des euklidischen Algorithmus an, der sich diese Erkenntnis zu Nutze macht.

Lösung

```
public static int euclid(int m, int n) {  
    if(n == 0) {  
        return m;  
    } else {  
        return euclid(n, m % n);  
    }  
}
```