

13.8 Übungsblatt 8



Bild 13.4: Aus dem Film: "Und täglich grüßt das Murmeltier ...", einem irgendwie rekursiven Film. (Foto: Wikimedia Commons)

Dieses Blatt wendet sich den Methoden und insbesondere der *Rekursion* zu.

Aufgabe 8.1: Lesen

Lesen Sie sich zunächst die folgenden Kapitel im [Javabuch] durch:

- 3.5.2 Setzen der Umgebungsvariablen PATH
- 7.7.1 Der cast-Operator
- 8.2.3 Mehrfache Alternative – switch
- 9.2 Methodendefinition und -aufruf
- 9.7 Iteration und Rekursion

Sie sollten Fragen zu diesen Inhalten beantworten können.

13.8.1 Methoden selbst schreiben

Legen Sie eine neue Klasse `MethodenUebung` mit einer `main`-Methode an. Danach implementieren Sie bitte in dieser Klasse die folgenden Methoden:

Aufgabe 8.2: Einfache arithmetische Methoden

1. Betrag einer beliebigen Zahl.
2. Eine Zahl aufrunden zur ganzen Zahl.
3. Ebenso abrunden zur ganzen Zahl.
4. Bestimmen, ob eine Zahl restlos durch eine andere Zahl teilbar ist.
5. Euklidische Distanz: Berechnen Sie den Abstand zweier beliebiger Punkte $P1(x1, y1)$ und $P2(x2, y2)$.
6. Bestimmen Sie die kleinste Zahl dreier beliebigen Zahlen.

Machen Sie sich bitte bei jeder benötigten Methode Gedanken über die Typen der Methodenparameter sowie des Rückgabetyps. Testen Sie Ihre entworfenen Methoden ausgiebig.

13.8.2 Methoden analysieren

Versuchen Sie bitte, die folgenden Aufgaben zunächst *ohne* Ausprobieren am Rechner zu lösen. Entwickeln Sie also bitte zuerst einen Entwurf Ihrer Lösung von Hand, bevor Sie diese am Rechner testen.

Aufgabe 8.3: Analyse verzweigter Methodenaufrufe, Aufrufbaum

Gegeben sei folgende Klasse:

```
class CallTree{
    public static int f() {System.out.print("f() "); g(); h(); return 1;}
    public static int g() {System.out.print("g() "); return 2;}
    public static void h() {System.out.print("h() "); }
    public static void b() {System.out.print("b() ");f(); h();}
    public static void main(String[] args) { b(); }
}
```

1. Analysieren Sie bitte den Code und überlegen Sie sich, was durch den Aufruf der Methode `b()` in der `main`-Methode ausgegeben wird.
2. Zeichnen Sie bitte den *Aufrufbaum*, der durch den Aufruf von `b()` entsteht.

Aufgabe 8.4: Vermeiden von mehreren Methodenausgängen

Gegeben sei folgender Code:

```
enum KartenFarbe {
    KARO, HERZ, PIK, KREUZ
}

static int farbenWert(KartenFarbe farbe) {
    if (farbe == KartenFarbe.KARO)
        return 9;
    else if (farbe == KartenFarbe.HERZ)
        return 10;
    else if (farbe == KartenFarbe.PIK)
        return 11;
    else
        return 12;
}
```

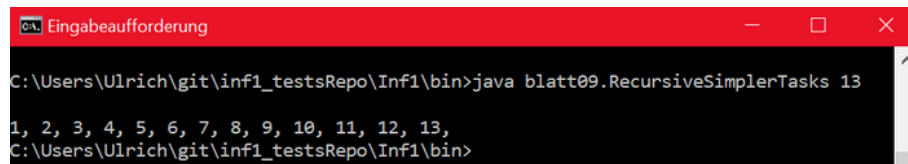
An diesem Code ist zu bemängeln, dass die Methode `farbenWert` an vielen Stellen verlassen wird. Ändern Sie bitte die Methode `farbenWert` so ab, dass Sie nur noch an ihrem Ende verlassen und eine `switch`-Anweisung eingesetzt wird.

13.8.3 Nichtverzweigende Rekursion

Aufgabe 8.5: Rekursion für die Folge 1, ..., n,

1. Schreiben Sie bitte eine rekursive Methode `prt1234(long n)`, welche die Zahlen 1, 2, ..., n, am Bildschirm ausgibt.

2. Holen Sie sich zum Demonstrieren Ihrer Lösung dieses n bitte als *Aufrufparameter* der `main`-Methode Ihrer Testklasse ab.
3. Starten Sie das Programm bitte auch über die Eingabeaufforderung (`cmd.exe`) unter Windows. Das sollte in etwa so aussehen (unter Windows):



```

C:\Users\Ulrich\git\inf1_testsRepo\Inf1\bin>java blatt09.RecursiveSimplerTasks 13
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
C:\Users\Ulrich\git\inf1_testsRepo\Inf1\bin>

```

Aufgabe 8.6: Rekursion für die Folge 1, 4, 9, ..., n^2 ,

Schreiben Sie bitte eine rekursive Methode `prtSqr1234(long n)`, welche die Zahlen 1, 4, 9, ..., n^2 , also die Quadratzahlen, am Bildschirm ausgibt.

Wie hoch, in Abhängigkeit von der größten ausgedruckten Zahl n^2 , ist die Rekursionstiefe Ihrer Methode?

Aufgabe 8.7: Rekursion für die Folge 2, 4, 6, ..., n ,

Schreiben Sie bitte eine rekursive Methode `prt2468(long n)`, welche die geraden Zahlen 2, 4, 6, ..., n , ausgibt. Falls n ungerade ist, soll als letzte Zahl die größte gerade Zahl m mit $m < n$ ausgegeben werden.

Aufgabe 8.8: Rekursion in Schleife übertragen

Gegeben sei folgendes Programmstück:

```

void xxx(long n) {
    if(n > 0L) {
        System.out.println(n);
        xxx(n-1);
    }
    else {}
}

```

- Überlegen Sie sich bitte die Ausgabe des Programms.
- Kommentieren Sie sich bitte in den Code ein, wo sich die *Rekursionsbasis* und die *Rekursionsfortsetzung* befindet.
- Schreiben Sie bitte eine iterative Variante der Methode `xxx`. Führen Sie dabei bitte keine neuen Variablen ein, d.h. verwenden Sie lediglich den Parameter n in Ihrer Iteration.

Aufgabe 8.9: Zahl rekursiv in andere Basis wandeln

Gegeben sei die Klasse `BasisWandler` in der Datei `BasisWandler.java` in ILIAS.

Dort ist die schon in der Vorlesung angesprochene Methode `zurBasisKausgeben` zu finden, die eine gegebene Dezimalzahl n in einer anderen Basis ausgibt. Wird zum Beispiel `zurBasisKausgeben(13, 2)` aufgerufen, so wird 1101 ausgegeben.

Wird allerdings zurBasisKausgeben(255, 16) aufgerufen, so wird 1515 ausgegeben. Das Problem ist, dass zweimal die Dezimalzahl 15 ausgegeben wird, nicht die Hexadezimalziffer 'F', wie es richtig wäre.

In dieser Aufgabe soll also entsprechend der Methode zurBasisKausgeben eine neue Methode String inBasisKwandeln(long n, int k) geschrieben werden, welche die Ziffern bei Basen $k > 10$ richtig darstellt.

1. Machen Sie sich bitte noch mal mit der Methode zurBasisKausgeben vertraut. Dazu eine Frage: Wenn $k == 2$ und $n == 255$ ist, welche Rekursionstiefe ist zu erwarten?
2. Hinweis: Die gegebene Methode intNachZiffer berechnet für eine gegebene Zahl die entsprechende Ziffer, für 15 also ein 'F'.
3. Vervollständigen Sie bitte die Methode String inBasisKwandeln(long n, int k), sodass die gewandelte Zahl mit den korrekten Ziffern in einer Zeichenkette ausgegeben wird.
4. Die main-Methode der Klasse enthält einen Testrahmen für diese Methode. Die Ausgabe sollte am Ende so aussehen:

42 zur Basis 10 ist	42 (OK)
13 zur Basis 2 ist	1101 (OK)
18 zur Basis 20 ist	I (OK)
399 zur Basis 20 ist	JJ (OK)
8000 zur Basis 20 ist	1000 (OK)
31 zur Basis 32 ist	V (OK)
1023 zur Basis 32 ist	VV (OK)
1024 zur Basis 32 ist	100 (OK)

Aufgabe 8.10: Zusatzaufgabe/Knobelei: Gegebene Zahl zur Basis k in Dezimalzahl wandeln

Der Dezimalwert d einer Zahl mit den n Ziffern $Z_1Z_2...Z_n$ zur Basis k lässt sich so berechnen (siehe Horner-Schema, z.B. in Wikipedia):

$$d(Z_1Z_2...Z_{n-1}Z_n) = \begin{cases} Z_1 & \text{falls } n = 1 \\ Z_n + k * d(Z_1Z_2...Z_{n-1}) & \text{sonst (d.h. } n > 1) \end{cases}$$

Beispiel: Die Hex-Zahl DEF ist so umgerechnet in dezimal:

$15 + 16 * d(DE) ==$

$15 + 16 * (14 + 16 * d(D)) ==$

$15 + 16 * (14 + 16 * 13) ==$

3567

Die Aufgaben im Einzelnen:

1. Schreiben Sie bitte eine Methode, die eine Ziffer in eine int-Zahl umrechnet:

```
/** Zeichen in int-Zahl wandeln
 *
 * @param c: Gegebene Ziffer aus 0 ... 9 und A ... Z
 * @return Dezimaler Wert der Ziffer
 */
int zifferNachInt(char c) {
    int ret;
    // TODO
    return ret;
}
```

2. Nutzen Sie diese Methode in der folgenden Methode `basisKinDezimal`, welche die obige mathematische Definition der rekursiven Funktion $d(Z_1Z_2...Z_n)$ implementiert:

```
/** Gegebene Zahl zur Basis k in Dezimalzahl umrechnen
 *
 * @param zahlBasisK: Zeichenkette mit Ziffern zur Basis k
 * @param k: Basis
 * @return umgerechnete Dezimalzahl
 */
long basisKinDezimal(String zahlBasisK, int k) {
    // TODO
}
```

3. Ergänzen Sie die `main`-Methode, sodass Ihre neue Funktion `basisKinDezimal` mit den gegebenen Testfällen getestet wird. Die Ausgabe sollte dann so aussehen:

```
42 zur Basis 10 ist dezimal ==      42 (OK)
1101 zur Basis 2 ist dezimal ==     13 (OK)
I zur Basis 20 ist dezimal ==       18 (OK)
JJ zur Basis 20 ist dezimal ==      399 (OK)
1000 zur Basis 20 ist dezimal ==    8000 (OK)
V zur Basis 32 ist dezimal ==        31 (OK)
VV zur Basis 32 ist dezimal ==     1023 (OK)
100 zur Basis 32 ist dezimal ==     1024 (OK)
```