Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Programmierung 1
WiSe 25/26
Moritz Balg
10.11.2025
bearbeiten bis: 16.11.2025

Übungsblatt 6

- Typanpassung und Methoden -

Aufgabe 1: Ausdrücke

Gegeben sind folgende Deklarationen:

```
boolean b ;
int i;
float k;
```

Füllen Sie folgende Tabelle aus. Für Ausdrücke in der ersten Spalte, die erlaubte Ausdrücke sind (in Spalte 2 = ja), geben Sie auch den Typ des Resultats an sowie den Ergebniswert. Für Ausdrücke, die nicht erlaubt sind, geben Sie an, weshalb nicht und überlegen sich, wie man ggfs. den Ausdruck sinnvoll erweitern könnte.

Mögliche Lösung:

Ausdruck	erlaubt?	Typ	Wert
`3 + 4L`	ja	long	7
`3L + 4`	ja	long	7
`3L + 4.`	ja	double	7.0
`3 + 4F`	ja	float	7.0
`3. + 4F`	ja	double	7.0
`i = 3 + 4F`	nein `i=(int)(3+4F)`	float, nach cast int	7
`i = (1L << 32) + 1L`	nein, Wert nach cast aber falsch weil Bits abgeschnitten `i = (int)((1L << 32) + 1L)`	long, nach cast int	1 (eigentlich 4294967297)
`b = (3 < 4D)`	ja	boolean	true
`b = 3 + 4F`	nein, auch nicht mit cast wegen starker Typisierung in Java		
`k = 3 + 4.0`	nein `k = (float)(3+4.0)`	double, nach cast float	7.0
`k = 3L + 4.0`	nein `k = (float)(3L+4.0)`	double, nach cast float	7.0

Aufgabe 2: Wertgleichheit

Erklären Sie, unter welcher Bedingung nach den Anweisungen

```
int x = ... ;
float f = x;
int y = (int) f;
```

davon ausgegangen werden kann, dass x==y gilt. Nennen Sie ggf. das kleinste Gegenbeispiel.

Mögliche Lösung:
Es darf bei der ersten Umwandlung keinen Rundungsfehler geben, was voraussetzt, dass x keine signifikanten Stellen enthält, die um einen Faktor größer als 2^{23} auseinanderliegen. Der kleinste Wert, für den dies nicht der Fall ist, ist $2^{24} + 1 = 16777217$, also weniger als 17 Millionen. Anmerkung: Hier ist spannend anzumerken, dass der diskutierte Fehler bei der impliziten, also vermeintlich sicheren Konvertierung in der zweiten Zeile auftritt, während (in diesem Fall) die explizite Konvertierung in der dritten Zeile unkritisch ist. Und auch, dass viele (was für welche?) Zahlen von ca. 16 Mio. bis ca. 2 Mia. von dem Fehler nicht betroffen sind.

Aufgabe 3: Inkrement

Analysieren Sie folgendes Programm und erläutern Sie präzise, wie es zu der Ausgabe kommt, die Sie nach Start des Programms sehen:

```
public class Erhoehen{
  public static void main (String [] args){
    int a = 5;
    a = a ++;
    System.out.println(a);
  }
}
```

Mögliche Lösung:
Der Ausdruck a++ bewirkt, dass der Wert der Variablen (5) um 1 erhöht wird (also 6) und dieser Wert (6) in der Variablen gespeichert wird. a++ ist ein Ausdruck, liefert also einen Wert. Der Wert dieses Ausdrucks ist der Wert der Variablen a vor der Erhöhung, also der Wert 5. Dieser Resultatwert des Ausdrucks wird, nachdem der Ausdruck komplett abgearbeitet ist und damit a den neuen Wert 6 erhalten hat, in der Variablen gespeichert, die auf der linken Seite der Zuweisung steht, also a. Damit bekommt a nun den Wert 5 zugewiesen, was der Inhalt der Variablen nach der Zuweisung ist.

Aufgabe 4: Typumwandlung

1. Gegeben sind folgende nacheinander in einem Programm stehende und noch unvollständige Programmstücke. Weiterhin ist eine erwartete Ausgabe angegeben, die zum vervollständigten Programmcode erwartet wird. Ergänzen Sie die Programmstücke, wo dies nötig ist (und nur dort), um Typanpassungen (cast-Operationen). Analysieren Sie dazu sorgfältig die Programmteile. Ausprobieren in eclipse hilft Ihnen nicht wirklich weiter.

Programmteil	geforderte Ausgabe
`int i = (1 << 8) + 1; byte b = i; System.out.println(b);`	1
`long l1 = b; System.out.println(l1);`	1
`float f = b + 0.5; System.out.println(f);`	1.5
`long l2 = f; System.out.println(l2);`	1

Mögliche Lösung:

```java
/**
 * Ergänzen, sodass die geforderte Ausgabe erscheint
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class CastErgaenzen {
    public static void main(String[] args) {
        int i = (1 << 8) + 1;
        byte b = (byte) i;
        System.out.println(b);

        long l1 = b;
        System.out.println(l1);

        float f = b + (float) 0.5;
        System.out.println(f);

        long l2 = (long) f;
        System.out.println(l2);
    }
}
```

2. Gegeben sei folgender Auszug aus einem Java-Programm:

```java
short s = -255;
System.out.println((byte)s);
```

Was wird auf dem Bildschirm ausgegeben? Erklären Sie, wie es zu der Ausgabe kommt.

Mögliche Lösung:

Es wird 1 ausgegeben, weil im Datentyp `short` ist $-255_{10} = 1111111100000001^*_2$ (Achtung Zweierkomplement!). Durch den cast werden die höherwertigen acht Bits abgeschnitten übrig bleibt 00000001^*_2. Hierdurch entsteht ein Vorzeichenwechsel.

Aufgabe 5: Palindromzahl 👾

Achtung

- Verwenden Sie in dieser gesamten Aufgabe keine String-Operationen, sondern arbeiten Sie nur mit ganzzahligen Werten und den darauf definierten Operationen!
- Sie können davon ausgehen, dass überall nur zulässige Werte vorkommen, die zudem alle positiv sind.

Für eine Zahl mit der Dezimaldarstellung $x_1x_2...x_{n-1}x_n$ ist deren Spiegelbild die Zahl mit der Darstellung $x_nx_{n-1}...x_2x_1$. Beispiel: Zu 1234 wäre das Spiegelbild die Zahl 4321. Eine Palindromzahl ist eine Zahl, die von vorne und von hinten gelesen den gleichen Wert hat, also gleich ihrem Spiegelbild ist. Beispiele: 191 und 23432 und 1221 sind Palindromzahlen, 123 ist es nicht.

Schreiben Sie ein Java-Programm `Palindromzahl`, das in `main` von der Kommandozeile eine positive `int`-Zahl `n` einliest. Anschließend soll der Spiegelwert von `n` auf dem Bildschirm in einer Zeile ausgegeben werden. In einer weiteren Zeile soll dann die Summe von `n` mit dem Spiegelwert ausgegeben werden. Nachfolgend wird dann folgender Algorithmus auf die Zahl `n` anwendet: solange `n` keine Palindromzahl ist, addiere zu `n` den Spiegelwert. Wenn diese Iteration abbricht, so gebe den aktuellen `n`-Wert auf dem Bildschirm in einer weiteren Zeile aus. Beispiele zu diesem Algorithmus:

- Für `n = 121` wäre der Ablauf wie folgt: `121` ist eine Palindromzahl, also bricht die Iteration sofort ab und `121` wird auf dem Bildschirm ausgegeben.
- Für n = 123 wäre der Ablauf wie folgt: `123` ist keine Palindromzahl. Also addiere dazu die Spiegelzahl: `123 + 321 = 444`. Die Überprüfung von `444` liefert, dass dies eine Palindromzahl ist, also bricht die Iteration an der Stelle ab und 444 wird ausgegeben.

Insgesamt werden also drei Zeilen auf dem Bildschirm ausgegeben!

Schreiben Sie folgende Methoden, um dieses Programm zu strukturieren:

1. Geben Sie eine Java-Methode `public static int spiegeln(int zahl)` an, die zu einem positiven `int`-Wert die Zahl erzeugt, die dem Spiegebild des Ursprungswertes entspricht. Hinweise:
   - Nutzen Sie die ganzzahlige Divison und Modulo-Bildung sowie Multiplikation mit 10.
   - Überlegen Sie sich mit Hilfe eines Blattes Papier und einer kleiner Beispielzahl, was Sie an Operationen machen müssen.
   - Überlegen Sie, wie eine Anweisung der Form `x = x * y + z` für geeignete `x, y, z` Ihnen helfen kann.
2. Geben Sie eine Java-Methode `public static int spiegelAddieren(int zahl)` an, die für eine int-Zahl deren Spiegelzahl erzeugt, die Ursprungs- und deren berechnete Spiegelzahl addiert und diese Summe der beiden Zahlen als Ergebnis der Methode liefert. Beispiel: zu 123 ist das Ergebnis $123 + 321 = 444$.
3. Geben Sie eine weitere Java-Methode `public static boolean palindromTest(int zahl)` an, die für eine `int`-Zahl feststellt, ob diese Zahl eine Palindromzahl ist, d.h. ob diese Zahl in Zifferndarstellung von vorne wie von hinten gelesen die gleiche Ziffernfolge hat. Verwenden Sie dazu die Methode `spiegeln` aus der früheren Teilaufgabe.

Beispiel

```
> java Palindromzahl 123
```

Ausgabe:

```
321
444
444
```

Beispiel

```
> java Palindromzahl 651
```

Ausgabe:

```
156
807
6666
```

Aufgabe 6: IBAN berechnen 👾

Achtung

Diese Aufgabe ist anspruchsvoller und nutzt kombiniert verschiedene Aspekte des bisher vermittelten Stoffs. Bearbeiten Sie zuerst die vorher angeführten einführenden Aufgaben zu Methoden, bevor Sie diese Aufgabe angehen. Beherrschen Sie die Komplexität dieser Aufgabe, indem Sie die Lösung strukturiert angehen (siehe Vorgabe unten). Überlegen Sie hier selbst wie Sie Ihr Program am besten mit Methoden strukturieren!

Die im SEPA-Verfahren genutzten IBAN-Kontonummern sind europaweit gültig. Für deutsche Konten sind die entsprechenden IBAN-Nummern nach einem festen Schema aufgebaut: Ein Ländercode aus zwei Buchstaben (für Deutschland DE), eine zweistellige Prüfzahl, eine achtstellige Bankleitzahl und eine zehnstellige Kontonummer. Es werden nachfolgende Beispiele zu folgenden Basiwerten angegeben: Ländercode=DE, Bankleitzahl=12345678, Kontonummer=123456 Die Berechnung einer solchen deutschen IBAN-Nummer geschieht in mehreren Schritten:

1. Zuerst wird der Ländercode normalisiert, so dass er nur aus Großbuchstaben besteht. Beispiel: aus de würde DE.
2. Derzeit vergebene Kontonummern können auch weniger als 10 Ziffern enthalten. In einem weiteren Schritt normalisiert man eine Kontonummer, indem fehlende Ziffern durch führende Nullen ergänzt werden, so dass die Kontonummer genau 10-stellig ist. Beispiel: aus 123456 wird 0000123456.
3. Dann wird eine BBAN erzeugt, die aus der Bankleitzahl gefolgt von der normalisierten Kontonummer gebildet wird. Diese ist also genau 18-stellig. Für die Beispielwerte gilt BBAN=123456780000123456.
4. Für die beiden (Groß-)Buchstaben des Ländercodes wird getrennt voneinander folgende Umwandlung durchgeführt. Der Großbuchstabe wird in einen Zahlencode umgewandelt, der der Position des Buchstabens im Alphabet entspricht (beginnend bei 1), und darauf 9 addiert. Beispiel: aus A wird 9+1=10, aus D wird 9+4=13 und aus E wird 9+5=14. Diese erzeugte Zahl ist also auf jeden Fall zweistellig, zwischen 10 für A und 35 für Z. An die BBAN wird dann zuerst der Zahlencode des ersten Buchstabens und dann der Zahlencode des zweiten Buchstabens hinten angehängt sowie zwei weitere Nullen. Im Beispiel entsteht daraus insgesamt 123456780000123456131400.
5. Danach wird die so erzeugte 24-stellige Ziffernfolge als Zahl aufgefasst und modulo 97 genommen. Eine 24-stellige Dezimalzahl sprengt allerdings den Wertebereich von int und sogar long. Deshalb bedient man sich eines Tricks (ohne Herleitung/Begründung): man nimmt nur die ersten 9 Ziffern, fasst diese als 9-stellige Zahl auf (welcher Java-Typ ist dafür geeignet?) und berechnet dazu den modulo-97 Wert. Diesen (ein- oder zweistelligen) Modulo-Wert ügt man vorne an die verbliebene 15-stellige Zahl an, nimmt davon wiederum die ersten 9 Ziffern, bildet davon den Modulo-97 Wert, hängt diesen wiederum vorne an den verbliebenen Rest usw. Dieses Verfahren bricht ab, wenn nur noch der Modulo-Wert aber keine restlichen Ziffern mehr vohanden sind. Beispiel zu 123456780000123456131400: 123456780 modulo 97 = 30, 300001234 modulo 97 = 22, 225613140 modulo 97 = 64, 640 modulo 97 = 58. Also ist 123456780000123456131400 modulo 97 = 58.
6. Dann zieht man von der Konstanten 98 den Modulo-97-Wert ab. Ist die resultierende Zahl kleiner als 10, so fügt man eine führende Null hinzu, so dass man auf jeden Fall eine zweistellige Zahl hat. Diese Zahl ist die Prüfzahl. Im Beispiel: 98-58=40.

7. Die IBAN-Nummer ist dann die normalisierte Länderkennung, gefolgt von den zwei Ziffern der Prüfzahl, gefolgti von der BBAN. Im Beispiel: DE40123456780000123456.

Geben Sie zu den einzelnen Schritten oder zu sinnvoll zusammengefassten Teilschritten jeweils eine Methode an, zu der Sie sich auch eine sinnvolle Signatur überlegen. Überlegen Sie sich sehr genau, wo welcher Datentyp angebracht ist und mit welchen Operationen Sie arbeiten. Lassen Sie sich zu Testzwecken (nicht in der Abgabeversion!) die Ergebnisse der Zwischenschritte ausgeben und überprüfen diese mit den Referenzwerten zum obigen Beispiel. Geben Sie zuletzt eine Methode `public static String erzeugeIban(String laenderkennung, String blz, String nummer)` an, die aus den gegebenen Einzelangaben eine korrekte IBAN-Kontonummer erzeugt. Geben Sie dann ein `main`-Methode in einer Klasse `IbanBerechnen` an, die 3 Argument zur Laufzeit über die Tastatur einliest: eine Länderkennung aus genau zwei Buchstaben, eine Bankleitzahl aus genau acht Ziffern und eine maximal 10-stellige Kontonummer. Lesen Sie diese Eingabewerte jeweils als `String` ein mit `sc.next()` für einen `Scanner` mit Namen `sc`. Bilden Sie aus den Eingabewerten die IBAN-Nummer und geben nur diese aus, gefolgt von einem Zeilenende.

Beispiel

Eingabe:
```
de 12345678 123456
```

Ausgabe:
```
DE40123456780000123456
```

Info

Sie können zu eigenen Testdaten ihr IBAN-Ergebnis überprüfen über https://www.iban.de/iban-berechnen.html

Aufgabe 7: Signum Methode

Geben Sie für alle numerischen Typen (außer char) jeweils eine Java-Methode an, die die Signum-Funktion berechnet. Die Signum-Funktion ist in der Mathematik definiert durch: signum : $\mathbb{R} \to \mathbb{R}$

$$\text{signum}(x) = \begin{cases} +1 \text{ falls } x > 0 \\ 0 \text{ falls } x = 0 \\ -1 \text{ falls } x < 0. \end{cases}$$

Der Resultattyp einer Methode sollte dabei vom gleichen Typ sein wie der Argumenttyp. Rufen Sie im Hauptprogramm (`main`) all ihre Methoden der Reihe nach auf mit den Werten -1, 0 und 1 (im jeweiligen Typ) und geben Sie das Ergebnis des Methodenaufrufs auf dem Bildschirm aus.

Info

- Seien Sie sorgsam darin, dass Sie durch die richtige Angabe des Argumenttyps auch die richtige Methode aufrufen.
- Sie überladen damit den Methodennamen signum.
- Überlegen Sie sich, wie man sinnvoll in main die Methodenaufrufe mit −1, 0 und 1 angeben kann.

Mögliche Lösung:

```java
/**
 * Signum-Funktion überladen
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Signum {
  public static byte signum(byte b) {
```

```
    // der korrekte Wert bleibt erhalten beim cast auf byte
    return (byte) ((b == 0) ? 0 : ((b < 0) ? -1 : 1));
  }

  public static short signum(short b) {
    // der korrekte Wert bleibt erhalten beim cast auf byte
    return (short) ((b == 0) ? 0 : ((b < 0) ? -1 : 1));
  }

  public static int signum(int b) {
    return (b == 0) ? 0 : ((b < 0) ? -1 : 1);
  }

  public static long signum(long b) {
    // alternativ wie oben möglich
    return (b == 0L) ? 0L : ((b < 0L) ? -1L : 1L);
  }

  public static float signum(float b) {
    return (b == 0f) ? 0f : ((b < 0f) ? -1f : 1f);
  }

  public static double signum(double b) {
    return (b == 0d) ? 0d : ((b < 0d) ? -1d : 1d);
  }

   // Testen des Signums-Funktions für verschiedene Datentypen
   public static void main(String[] args) {
     for (int i = -1; i <= +1; i++) {
       System.out.println("Wert ist " + i);
       System.out.println("byte: " + signum((byte)i));
       System.out.println("short: " + signum((short)i));
       System.out.println("int: " + signum(i));
       System.out.println("long: " + signum((long)i));
       System.out.println("float: " + signum((float)i));
       System.out.println("double: " + signum((double)i));
     }
   }
}
```

Aufgabe 8: ggt mit Modulo

Schreiben Sie ein Programm, das eine oder mehr natürliche Dezimalzahlen von der Kommandozeile liest und deren größten gemeinsamen Teiler ausgibt. Schreiben und verwenden Sie dazu eine Methode ggT, die den größten gemeinsamen Teiler zweier natürlicher Zahlen berechnet.

Außer der in der Vorlesung vorgestellten Originalversion des Euklidischen Algorithmus gibt es dafür folgende verbesserte Version mittels der Modulo-Operation:

Anstatt dass in jedem Rechenschritt die kleinere Zahl von der größeren abgezogen wird, wird eine der beiden Zahlen durch den Rest bei ihrer Division durch die andere Zahl ersetzt. Dadurch wird letztere die grössere der beiden Zahlen! Wieder gilt: Wenn die kleinere der beiden Zahlen 0 ist, ist die größere das Ergebnis.

Wieso ist diese Version äquivalent zur originalen? Wieso ist sie besser? Implementieren Sie die Methode, möglichst effizient, in folgenden Varianten:

1. als Originalversion mittels Schleife
2. als Originalversion mittels Rekursion

3. als Moduloversion mittels Schleife
4. als Moduloversion mittels Rekursion

Gestalten Sie die rekursiven Methoden endrekursiv, falls möglich. Falls nicht, geben Sie hierfür eine Begründung an.

Info

Um Spezialfälle in einer rekursiven Lösung zu behandeln, muss man ggf. eine nichtrekursive "Einstiegsmethode" und eine rekursive "Arbeitsmethode" bereitstellen. Die erste wird aufgerufen und bearbeitet die Spezialfälle oder gibt den Aufruf weiter an die zweite, die den Regelfall bearbeitet.

Mögliche Lösung:

```
public class GGT2 {
  public static void main(String[] args) {
    int a = Integer.parseInt(args[0]);
    for (int i = 1; i < args.length; ++i) {
      int b = Integer.parseInt(args[i]);
      a = ggTMinusS(a, b);
    }
    System.out.println(a);
  }

  public static int ggTMinusS(int a, int b) {
    if (a == 0) {
      return b;
    } else {
      while (b != 0) {
        if (a > b) {
          a = a - b;
        } else {
          b = b - a;
        }
      }
      return a;
    }
  }

  public static int ggTMinusR(int a, int b) {
    if (a == 0) {
      return b;
      // Alternative Implementierung
      //return ggTMinusRrekursiv(a, b);
      // Alternative Implementierung
      //return ggtminusrekursiv(a,b);
    }
    public static int ggtminusrekursiv(int a,int b){
      if(b==0){
        return a ;
      }else{
        if(a>b){
          a=a-b ;
        }else{
          b=b-a ;
        }
        return ggtminusrekursiv(a,b );
      }
    }
```

```
    public static int ggTModuloS(int a, int b) {
      while (b != 0) {
        int aa = a;
        a = b;
        b = aa % b;
      }
      return a;
    }

    public static int ggTModuloR(int a, int b) {
      if (b == 0) {
        return a;
      } else {
        return ggTModuloR(b, Math.abs(a % b));
      }
    }
}
```

Aufgabe 9: Zifferzählen

Implementieren Sie eine **rekursive** Methode `public static int countN(int n, int num)`. Diese Methode soll zählen wie oft die Ziffer `n` in der Zahl `num` vorkommt.

Beispiel

- `countN(7, 70701277)` gibt den Wert `4` zurück.
- `countN(0, 70701277)` gibt den Wert `2` zurück.
- `countN(7, 111)` gibt den Wert `0` zurück.

Info

- Nutzen Sie keine Schleifen.
- Nutzen Sie geeignete Operationen auf ganzen Zahlen.
- Der Bedingungsausdruck (?-Operator) kann hier nützlich sein.
- Sie können davon ausgehen, dass nur gültige Werte übergeben werden.

Mögliche Lösung:

```
public class Demo {
  public static void main(String[] args) {
    System.out.println(countN(7, 70_701_277));
  }

  public static int countN(int n, int num) {
    if(num < 10) return (num != 0 && num%n == 0)?1:0;
    int last = num % 10;
    return ((last != 0 && last%n == 0)?1:0) + countN(n, num/10);
  }
}
```