Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences
Fachbereich Informatik
Prof. Dr. Rudolf Berrendorf

Einführung in die Programmierung
WS 19/20
Ausgabe : 04.11.2019
Bearbeiten bis : 10.11.2019

Übung 6

Aufgabe 1 :
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 (Hinweis: Cast).

Ausdruck	erlaubt?	Typ (anzugeben)	Wert (anzugeben)
`3 + 4L`			
`3L + 4`			
`3L + 4.`			
`3 + 4F`			
`3. + 4F`			
`i = 3 + 4F`			
`i = (1L << 32) + 1L`			
`b = (3 < 4D)`			
`b = 3 + 4F`			
`k = 3 + 4.0`			
`k = 3L + 4.0`			

Aufgabe 2 :
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 Programmstücke. Ausprobieren in eclipse hilft Ihnen nicht wirklich weiter.

Programmstück	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

Aufgabe 3 :
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.

Aufgabe 4 :
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);
  }
}
```

Aufgabe 5 :
Zum Jahreswechsel 2008 nach 2009 waren zahlreiche neuentwickelte Musikspielgeräte der Firma Microsoft kaum noch bedienbar, weil im Programmcode zur Berechnung des Datums ein Fehler vorhanden war (was man dann erst nachträglich analysiert hat). Hier ist der relevante Ausschnitt aus der Berechnung:

```
static int berechneJahr(int days) {
  int year = 1980; // 01.01.1980 ist der Tag 1

  while (days > 365) {
    if (isLeapYear(year)) {
      if (days > 366) {
        days = days - 366;
        year = year + 1;
      }
    } else {
      days = days - 365;
      year = year + 1;
    }
  }
  return year;
}
```

Die Methode bekommt als Eingabe eine Zahl, die die Anzahl der Tage seit dem 01.01.1980 darstellt, wobei der 01.01.1980 der Tag mit der Nummer 1 ist. Für den 01.01.2009 ist die Zahl 10593. (Hinweis: Der Fehler tritt nicht nur beim Wechsel nach 2009 auf.) Aus dieser Tagesangabe soll die Jahreszahl berechnet werden, indem man solange von der Tagesanzahl 365 für Nichtschaltjahre beziehungsweise 366 für Schaltjahre abzieht, bis die Anzahl der verbliebenen Tage kleiner als 366 ist. Die im Programm verwendete Methode **`isLeapYear(int year)`** liefert **`true`**, wenn die Jahreszahl ein Schaltjahr ist und ansonsten **`false`**.

1. Was wurde in dem angegebenen Algorithmus falsch gemacht? Wie kann man den Fehler beheben? Sie sollen hierbei kein komplett neues Verfahren entwickeln, sondern den existierenden Code durch möglichst wenige Änderungen korrigieren. Lösen Sie erst *durch Nachdenken* diese Teilaufgabe, bevor Sie weitermachen. Nehmen Sie Beispielwerte (u.a. den problematischen Wert), vollziehen den Algorithmus mit diesen Werten nach, überlegen sich die Ursache eines möglicherweise dabei auftretenden Problems und finden Sie eine Lösung.
2. Realisieren Sie ihre Änderungen zur Korrektur des gegebenen Codes.
3. Geben Sie eine eigene Methode **`isLeapYear(int year)`** an, die **`true`** oder **`false`** liefert, je nachdem ob die Jahreszahl zu einem Schaltjahr gehört oder nicht. Sie können davon ausgehen, dass das Argument größer als 1582 ist.
4. Testen Sie obige Methode **`berechneJahr`** mit verschiedenen Tageszahlen, darunter auch die für den 01.01.2009 (entspricht 10594).

Aufgabe 6 :
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.

Hinweis: 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.

Aufgabe 7 :
Schreiben Sie ein Programm, das zu drei Werten x, y, ε aus der Kommandozeile mit $x, y, \varepsilon \in \mathbb{R}, x < y, \varepsilon > 0, |y - x| > \varepsilon$ zwei verschiedene Zufallfszahlen aus dem Intervall $[x, y)$ erzeugt und auf dem Bildschirm ausgibt. Zwei Zahlen a, b sollen dann als verschieden gelten, wenn $|a - b| \geq \varepsilon$.

Hinweise:

- Die Methode `Math.random()` liefert mit jedem Aufruf eine pseudozufällige Zahl aus dem Intervall $[0, 1)$.
- Es ist unwahrscheinlich, aber nicht ausgeschlossen, dass zwei aufeinanderfolgende Aufrufe dieser Methode die gleiche Zufallszahl liefern. Sie müssen also solange zwei zulässige Zahlen aus dem gewünschten Intervall erzeugen, bis diese die gewünschte Eigenschaft besitzen.

Aufgabe 8 (1 Punkt) :
Dies ist eine Praktomataufgabe!

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).

Die im neuen SEPA-Verfahren benutzten europaweit gültigen IBAN-Kontonummern haben einen vorgegebenen Aufbau: Ein Ländercode aus zwei Buchstaben (für Deutschland etwa 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 IBAN-Nummer geschieht in mehreren Schritten (in dieser hier beschriebenen etwas vereinfachten Form nur auf deutsche Kontenangaben anwendbar):

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 fügt 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, gefolgt 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.

Beispieleingabe:

```
de 12345678 123456
```

Ausgabe dazu:

```
DE40123456780000123456
```

Hinweis: Sie können zu eigenen Testdaten ihr IBAN-Ergebnis überprüfen über **`http://www.iban.de/iban-berechnen.html`**

Aufgabe 9 (1 Punkt) :
Dies ist eine Praktomataufgabe!

Hinweise / Vorgaben für diese Aufgabe:

- 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.

Aufgabenstellung:

1. Für eine Zahl mit der Dezimaldarstellung $x_1x_2...x_{n-1}x_n$ ist deren Spiegelbild die Zahl mit der Darstellung $x_nx_{n-1}...x_2x_1$. 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. Beispiel: Zu 1234 wäre das Spiegelbild die Zahl 4321. Hinweise:
   - Nutzen Sie die ganzzahlige Divison und Modulo-Bildung sowie Multiplikation mit 10.
   - 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. Beispiele: 191 und 23432 und 1221 sind Palindromzahlen, 123 ist es nicht. Verwenden Sie dazu die Methode `spiegeln` aus der früheren Teilaufgabe.
4. Geben Sie nun ein Java-Programm `Palindromzahl` an, dass in `main` von der Kommandozeile eine positive int-Zahl n einliest, dann den Spiegelwert von n auf dem Bildschirm ausgibt und dann in einer weiteren Zeile die Summe von n mit dem Spiegelwert ausgibt. Nutzen Sie dazu ihre Methoden.
   Nachfolgend wird dann folgender Algorithmus auf die Zahl n anwendet: solange n keine Palindromzahl ist, addiere zu n den Spiegelwert (über `spiegelAddieren`). Wenn diese Iteration abbricht, so gebe den aktuellen n-Wert auf dem Bildschirm aus.
   Beispiele zu diesem Vorgehen:
   - 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.

Beispiele für Programmläufe:

1. Für den Aufruf `java Palindromzahl 123` wäre die Ausgabe wie folgt:

```
321
444
444
```

2. Für den Aufruf `java Palindromzahl 651` wäre die Ausgabe wie folgt:

```
156
807
6666
```