252-0027-00: Einführung in die Programmierung

Übungsblatt 3

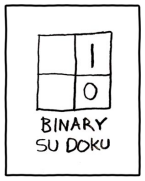
Abgabe: 17. Oktober 2023, 23:59

Checken Sie mit Eclipse die neue Vorlage aus, indem Sie wie in Übung 1 und 2 im Menü zuerst auf ihr Repository in der Repository-Ansicht rechtsklicken und *Pull* wählen. Danach Importieren Sie das neue Projekt durch Rechtsklick auf *uo3* und Auswahl von *Import Projects...*

Beachten Sie, dass Sie in dieser Übung mehrere unabhängige Programme im selben Eclipse-Projekt haben werden. Bevor Sie ein Programm starten, achten Sie deshalb darauf, dass Sie die richtige Datei im Package Explorer ausgewählt oder im Editor geöffnet haben. *Vergessen Sie nicht, Ihren Programmcode zu kommentieren!*

Aufgabe 1: Binärdarstellung

Schreiben Sie ein Programm "Binaer.java", das eine positive Zahl Z einliest und dann die Binärdarstellung druckt. (Hinweis: Finden Sie zuerst die grösste Zahl k, so dass die Zweierpotenz $K = 2^k$ kleiner als Z ist.)
Beispiel: Für $Z = 14$ sollte Ihr Programm `1110` ausgeben.



xkcd: Su Doku by Randall Munroe
(CC BY-NC 2.5)

Aufgabe 2: Grösster gemeinsamer Teiler

Schreiben Sie ein Programm "GGT.java", das den grössten gemeinsamen Teiler (ggT) zweier ganzer Zahlen mithilfe des Euklidischen Algorithmus berechnet. Hierbei handelt es sich um eine iterative Berechnung, die auf folgender Beobachtung basiert:

1. Wenn x grösser oder gleich y ist und sich x durch y teilen lässt, dann ist der ggT von x und y gleich y;
2. andernfalls ist der ggT von x und y der gleiche wie der ggT von y und $x \,\%\, y$.

Üben Sie diesen Algorithmus zuerst von Hand an ein paar Beispielen und schreiben Sie dann das Java Programm.

Beispiel: Für $x = 36$ und $y = 44$:

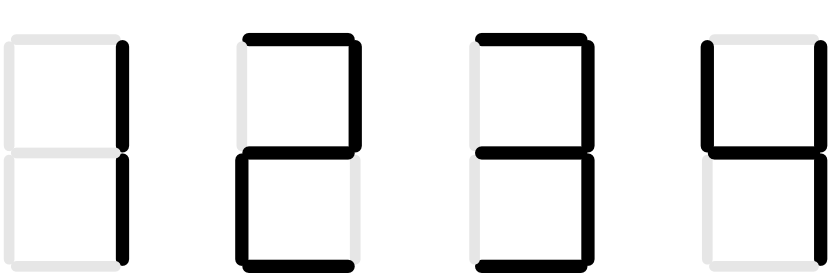
$$ggT(\underset{x}{36}, \underset{y}{44}) \overset{2.}{=} ggT(44, \underbrace{36 \,\%\, 44}_{36}) \overset{2.}{=} ggT(36, \underbrace{44 \,\%\, 36}_{8}) \overset{2.}{=} ggT(8, \underbrace{36 \,\%\, 8}_{4}) \overset{1.}{=} 4$$

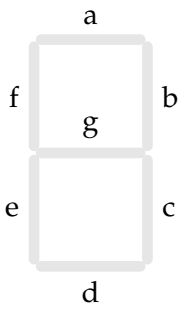
Aufgabe 3: Zahlenerkennung

Für diese Aufgabe verwenden wir einen String um die erleuchtenden Segmente einer Siebensegmentanzeige zu kodieren. Die Segmente sind, wie im Bild gezeigt, von a bis g nummeriert. Die Kodierung einer möglichen Anzeige ist ein String, in welchem der Buchstabe 'x' genau dann vorkommt, wenn das 'x'te Segment der Anzeige erleuchtet ist. Zum Beispiel wird die Zahl 2 kodiert durch 'abged'. Zur Einfachheit darf angenommen werden, dass kein Buchstabe mehr als einmal in der Kodierung vorkommt und dass nur die Zahlen 0 bis 9 kodiert werden.

Schreiben Sie ein Programm "Zahlen.java", das einen String, der eine Anzeige kodiert, einliest und die kodierte Zahl als Integer ausgibt. Überlegen Sie wie viele IF Blöcke benötigt werden um jede Zahl zu erkennen.

Tipp: Sie können `str.contains("a")` verwenden, um zu überprüfen, ob ein String `str` den Buchstaben 'a' enthält.





Aufgabe 4: Berechnungen

Implementieren Sie die Methode `Calculations.checksum(int x)`, das heisst die Methode `checksum` in der Klasse `Calculations`. Die Methode nimmt einen Integer `x` als Argument, welcher einen nicht-negativen Wert hat. Die Methode soll die Quersumme von `x` zurückgeben. Sie sollen für diese Aufgabe **keine** Schleife verwenden.

Beispiele

- `checksum(258)` gibt 15 zurück.
- `checksum(49)` gibt 13 zurück.
- `checksum(12)` gibt 3 zurück.

Hinweis: Für einen Integer `a` ist `a % 10` die letzte Ziffer und `a / 10` entfernt die letzte Ziffer. Zum Beispiel `258 % 10` ist 8 und `258 / 10` ist 25.

Aufgabe 5: Scrabble

In dieser Aufgabe sollen Sie Scrabble-Steine legen, mittels ASCII-Art auf der Konsole. Vervollständigen Sie die Methode `drawNameSquare` in der Klasse `Scrabble`. Diese Methode nimmt einen Namen als String-Parameter und soll den Namen als in einem Quadrat angeordnete Scrabble-Steine auf der Konsole (`System.out`) ausgeben. Wenn z.B. der String `Alfred` übergeben wird, sollte folgendes Bild ausgegeben werden:

```
+---+---+---+---+---+---+
| A | L | F | R | E | D |
+---+---+---+---+---+---+
| L |               | E |
+---+               +---+
| F |               | R |
+---+               +---+
| R |               | F |
+---+               +---+
| E |               | L |
+---+---+---+---+---+---+
| D | E | R | F | L | A |
+---+---+---+---+---+---+
```

Ihr Code braucht nur Namen der Länge 3 oder länger unterstützen. Für einen Namen mit Länge 3, z.B. `Jim`, sollte die Ausgabe so aussehen:

```
+---+---+---+
| J | I | M |
+---+---+---+
| I |   | I |
+---+---+---+
| M | I | J |
+---+---+---+
```

Beachten Sie, dass Ihr Programm keinerlei andere Ausgabe als das Scrabble-Quadrat machen darf.

Aufgabe 6: Weakest Precondition

Bitte geben Sie für die folgenden Programmsegmente die schwächste Vorbedingung (weakest precondition) an. Bitte verwenden Sie Java Syntax (die Klammern `{` und `}` können Sie weglassen). Alle Anweisungen sind Teil einer Java Methode. Alle Variablen sind vom Typ `int` und es gibt keinen Overflow.

1.
```
P: { ?? }
S:   m = n * 4;  k = m - 2;
Q: { n > 0 && k > 5 }
```

2.
```
P: { ?? }
S:   m = n * n;  k = m * 2;
Q: { k > 0 }
```

3.
```
P: { ?? }
S:   if (x > y) {
       z = x - y;
     } else {
       z = y - x;
     }
Q: { z > 0 }
```