

2. Übung

Abgabe bis 15.05.2017, 10:00 Uhr

Bitte beachten:

Generelle Hinweise zu den Aufgaben und deren Bearbeitung finden Sie auf dem „organisatorischen Übungsblatt“. Sie können die Aufgaben erst im EST abgeben, nachdem Sie in die Tafelübungen eingeteilt wurden (voraussichtlich Sonntag).

Einzelaufgabe 2.1: Bin2Long

10 EP

In dieser Aufgabe sollen Sie ein Programm ergänzen, das als Zeichenketten gegebene Binärzahlen in Zahlen vom Typ `long` umrechnet – beachten Sie dabei unbedingt, dass die Eingabe stets in entsprechender *Zweierkomplement-Darstellung* vorliegt, jedoch *beliebige Länge* haben kann (sofern vorhanden, ist das erste „Bit“ ein Hinweis auf das Vorzeichen)! Erstellen Sie eine Klasse `Bin2Long` und implementieren Sie die folgenden *statischen* Methoden.

- a) Die Methode `int check(String s)` soll testen, ob eine gültige Zeichenkette übergeben wurde und je nach vorliegendem Fall folgendes zurückgeben:

Rückgabe	Ergebnis der Prüfung
-3	falls die übergebene Zeichenkette <code>s</code> <code>null</code> ist
-2	falls <code>s</code> leer ist
-1	falls <code>s</code> eine legitime Binärzahl enthält (unabhängig vom Wertebereich!)
<code>pos</code>	die <i>position</i> des ersten ungültigen Zeichens in <code>s</code> (0-indiziert und positiv!)

- b) Die Methode `long convertBit(char b)` rechnet ein einzelnes Bit in die zugehörige Zahl um. Sie dürfen hier ohne Prüfung annehmen, dass `b` gültig (also nur `'0'` oder `'1'`) ist.
- c) Die Methode `long convert(String s)` führt die eigentliche Umrechnung der gesamten Zeichenkette durch. Falls die Eingabe ungültig ist oder es zu einem Überlauf käme, dann soll Ihre Methode den Wert `0L` zurückgeben.

Außer den Methoden der `String`-Klasse, die dem *Abruf* einzelner Zeichen oder der *Längenbestimmung* der Zeichenkette dienen, dürfen Sie **keine** anderen Java-Bibliotheksmethoden verwenden! Geben Sie Ihre Lösung als `Bin2Long.java` über EST ab.

Einzelaufgabe 2.2: Rochambeau

11 EP

Sicher kennen Sie noch das Spiel **Schere, Stein, Papier**. In dieser Aufgabe betrachten wir eine erweiterte Variante mit „**fünf Gesten**“. Folgende Tabelle zeigt, welche Gesten einander schlagen:

		<i>Stein</i> (A)	<i>Schere</i> (B)	<i>Papier</i> (C)	<i>Brunnen</i> (D)	<i>Streichholz</i> (E)
	<i>Stein</i> (A)	0	+	–	–	+
	<i>Schere</i> (B)	–	0	+	–	+
	<i>Papier</i> (C)	+	–	0	+	–
	<i>Brunnen</i> (D)	+	+	–	0	–
	<i>Streichholz</i> (E)	–	–	+	+	0

Beispiel (erste Zeile): Stein macht Schere und Streichholz kaputt (d.h. A gewinnt gegen B und E), wird aber vom Papier eingewickelt und versinkt im Brunnen (d.h. A verliert gegen C und D).

Angenommen, zwei Spieler \mathcal{X} und \mathcal{Y} spielen mehrere Runden gegeneinander, z.B.:

	Runde 1	Runde 2	Runde 3	...
Spieler \mathcal{X}	B	D	C	...
Spieler \mathcal{Y}	A	E	C	...

Dabei notieren sie mit den entsprechenden Buchstabenkürzeln, welche Geste sie jeweils gewählt haben, im Beispiel also „BADECC...“. Nach den ersten drei Runden hat Spieler \mathcal{Y} gewonnen und es steht 0 : 2. In dieser Aufgabe sollen Sie eine Klasse `Rochambeau` mit den folgenden Methoden implementieren, die die notierte Folge von Gesten auswertet und den Spielstand ermittelt:

- a) Die Methode `int check(String gestures)` soll die notierte Gestenfolge untersuchen und folgende Rückgabewerte (Priorität von oben nach unten) liefern:

Rückgabe	Ergebnis der Prüfung
–3	falls die übergebene Zeichenkette <code>gestures</code> leer oder null ist
–2	falls die Anzahl der Gesten in <code>gestures</code> ungerade ist
–1	falls unerlaubte Gesten in <code>gestures</code> vorkommen (erlaubt sind ausschließlich die Großbuchstaben A, B, C, D, E)
0	sonst

- b) Die Methode `int eval(char x, char y)` erhält die garantiert gültigen Gesten (x , y) der beiden Spieler \mathcal{X} bzw. \mathcal{Y} und soll folgende Rückgabewerte liefern:

Rückgabe	Ergebnis der Prüfung
–1	Spieler \mathcal{X} verliert diese Runde
0	die Runde ist unentschieden, weil beide die gleiche Geste gewählt haben
1	Spieler \mathcal{X} gewinnt diese Runde

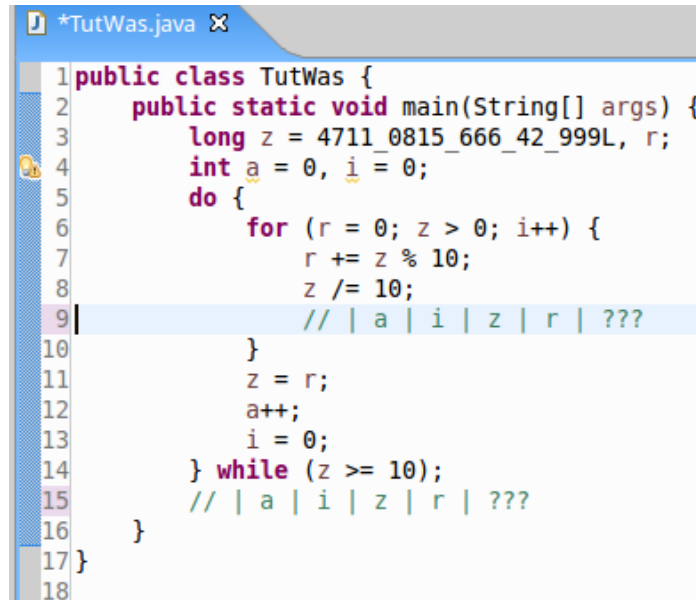
- c) Die Methode `int[] decide(String gestures)` ermittelt den Spielstand am Ende der notierten Gestenfolge und gibt ihn als `int`-Feld mit zwei Einträgen zurück, wobei der erste Eintrag die Anzahl der von Spieler \mathcal{X} gewonnenen Runden darstellt, während der zweite entsprechend für Spieler \mathcal{Y} steht. Falls die Eingabe ungültig ist, dann soll die Methode ein `int[]`-Feld mit nur einem Eintrag zurückgeben, der den Fehlercode von `check` enthält.

Geben Sie Ihre Lösung als `Rochambeau.java` über EST ab.

Gruppenaufgabe 2.3: Schreibtischlauf

19 GP

Betrachten Sie folgendes Programm:



```

1 public class TutWas {
2     public static void main(String[] args) {
3         long z = 4711_0815_666_42_999L, r;
4         int a = 0, i = 0;
5         do {
6             for (r = 0; z > 0; i++) {
7                 r += z % 10;
8                 z /= 10;
9                 // | a | i | z | r | ???
10            }
11            z = r;
12            a++;
13            i = 0;
14        } while (z >= 10);
15        // | a | i | z | r | ???
16    }
17 }
18

```

- Zeichnen Sie den **Programmablaufplan (PAP)** der Methode `TutWas.main`.
- Führen Sie das Programm nun „gedanklich“ in einem sogenannten **Schreibtischlauf** aus. Geben Sie die Werte der genannten Variablen (`a`, `i`, `z`, `r`) an den mit „???“ markierten Stellen bei *jedem* Durchgang (mehrfache Ausführung möglich!) tabellarisch in folgender Form an:

<i>a</i>	<i>i</i>	<i>z</i>	<i>r</i>
0	0

- Was zählen die Variablen `a` und `i`?
- Was berechnet dieses Programm *im Allgemeinen* (also für beliebige Startwerte von `z`) in der Variablen `z` am Ende?

Geben Sie Ihre Lösung als `Schreibtischlauf.pdf` über das [EST](#) ab.

Gruppenaufgabe 2.4: Zeichengeometrie

20 GP

In dieser Aufgaben sollen Sie eine „Methodenbibliothek“ für ASCII-Geometrie entwickeln. Laden Sie dazu die Datei `Zeichengeometrie.java` von der Aud-Homepage und ergänzen Sie die Methoden nach den Anweisungen in den folgenden Teilaufgaben.

Bitte beachten Sie dabei folgenden wichtigen Zusammenhang: Die von Methode zu Methode weitergereichte `zeichenflaeche` ist ein `2D-char[][]`-Feld, dessen „Ecke `[0][0]`“ gedanklich „oben/links“ ist, während *alle* Methoden das aus der Geometrie typische Koordinatensystem verwenden, bei dem der „Ursprung `(0,0)`“ tatsächlich „unten/links“ ist, wie es [Abbildung 1](#) verdeutlicht. Bitte beachten Sie auch, dass die zu zeichnenden Objekte über die `zeichenflaeche` hinausragen *dürfen* – allerdings nur die innerhalb der `zeichenflaeche` liegenden Teile davon werden tatsächlich gezeichnet und dargestellt (z.B. mittels `aufBildschirmAusgeben`).

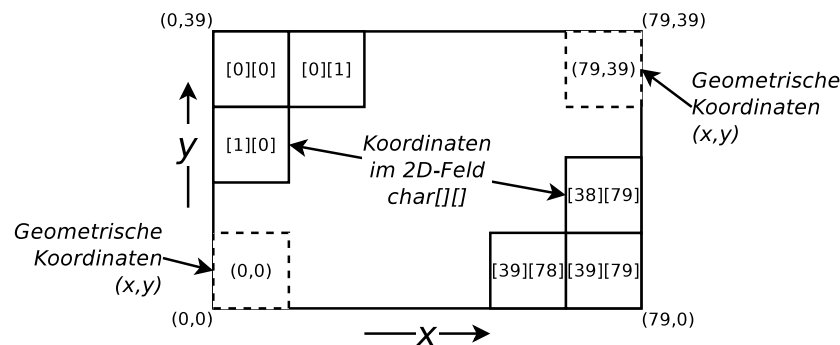


Abbildung 1: Zusammenhang zwischen 2D-char[] []-Feld und Geometrie

- Die Methode `erzeugeNeueZeichenflaeche` erzeugt ein 2D-char[] []-Feld mit `hoehe` Zeilen und `breite` Spalten an. Anschließend initialisiert die Methode *jeden einzelnen* Eintrag im Feld mit einem **Leerzeichen** (Space bzw. ' ' bzw. ASCII-Zeichen mit Code 32₍₁₀₎!) und gibt das Feld zurück.
- Die Methode `zeichnePunkt` erhält die `zeichenflaeche`, die geometrischen Koordinaten (x, y) sowie einen „Zeichenstift“. Sie ersetzt den zu (x, y) entsprechenden Eintrag im Feld `zeichenflaeche` mit dem `zeichen` und gibt die veränderte `zeichenflaeche` zurück. Falls die Koordinaten (x, y) außerhalb der `zeichenflaeche` liegen, dann muss diese Methode die unveränderte `zeichenflaeche` zurückgeben.
- Die Methode `zeichneLinie` soll analog zu `zeichnePunkt` eine ganze Linie zwischen $(startX, startY)$ und $(endeX, endeY)$ zeichnen, wobei die Endpunkte der Linie immer gezeichnet werden müssen (auch wenn sie gleich sind und die Linie damit zu einem Punkt degeneriert). Bedenken Sie, dass die Linie horizontal, vertikal (!) oder beliebig diagonal verlaufen kann sowie dass sie über den Zeichenrand hinausgehen darf. Es ist auch nicht definiert, in welcher Lage Start- und Endpunkt zueinander liegen! Zeichnen Sie so genau wie möglich (Multiplikationen vor Divisionen!). Tipp: Sie dürfen `zeichnePunkt` aufrufen...
- Die Methode `zeichnePolygon` erhält im Feld `koordinaten` einen ganzen Linienzug $x_0, y_0, x_1, y_1, x_2, y_2, \dots$, d.h. je zwei aufeinander folgende Einträge in diesem Feld stellen einen Punkt (x_i, y_i) dar, der mit dem vorhergehenden mit einer Linie verbunden werden soll.
- Die Methode `zeichneRechteck` soll ein Rechteck mit den gegebenen Eckpunkten zeichnen. Sind die x - und/oder die y -Koordinaten gleich, dann degeneriert das Rechteck zu einer Linie oder sogar zu einem Punkt! Genau dann, wenn `ausgefuehlt == true` ist, soll die innere Fläche des Rechtecks mit dem gleichen „Zeichenstift“ `zeichen` befüllt werden.
- Die Methode `zeichneKreis` soll einen unausgefüllten Kreis mit Mittelpunkt (x_M, y_M) und Radius r so genau und vollständig wie möglich zeichnen. Verwenden Sie dabei folgenden Zusammenhang und runden Sie so spät wie möglich: $(x - x_M)^2 + (y - y_M)^2 = r^2$. Stellen Sie r^2 in Java durch `r*r` dar; für die Quadratwurzel dürfen Sie `Math.sqrt` verwenden.

Geben Sie Ihre Lösung als `Zeichengeometrie.java` über EST ab.