Programmausführung auf Papier (fortgeschritten)

Wir betrachten zunächst folgende Methode:

```
1   public double arraymittelwert(int[] a) {
2       int sum;
3       sum = 0;
4       for(int i = 0; i < a.length; i++) {
5           sum = sum + a[i];
6       }
7       double mw;
8       mw = sum / a.length;
9       return mw;
10  }
```

Wir legen nun nach folgenden Regeln eine Tabelle an:

- Spalte 1 und 2 erhalten die Zählung für Schrittnummer bzw. für die aktuelle Programmzeile. Außerdem erhält jede Variable eine eigene Tabellenspalte.
- Ist eine Variable zu einem Zeitpunkt nicht angelegt, so streichen wir das Feld durch. Ist die Variable angelegt, aber der Wert unbekannt, so schreiben wir „?“
- Jede Programmzeile enthält eine eigene Tabellenzeile, auch wenn sich kein Wert ändert. Falls sich der Wert einer Variablen nicht ändert, so kann das betreffende Feld einfach leer gelassen werden.
- Eine Programmzeile nur mit „}“ kann ignoriert werden.

Für die Beispielmethode erhalten wir folgende Tabelle (Hinweis: wäre dies eine Klausuraufgabe, so wären die fettgedruckten Teile vorgegeben)

Schritt	**Prog.Zeile**	**a[0]**	**a[1]**	**a[2]**	sum	i	mw
1	**1**	**4**	**8**	**3**	----	----	----
2	2				?	----	----
3	3				0	----	----
4	4					0	----
5	5				4		----
6	4					1	----
7	5				12		----
8	4					2	----
9	5				15		----
10	4					3	----
11	7					----	?
12	8					----	5
13	9					----	

Programmausführung auf Papier (fortgeschritten) Teil 2

Mit dieser Art der „Programmausführung per Hand“ erreichen wir eine gute Nachvollziehbarkeit. Beispielsweise können wir an der Tabelle ablesen, dass wir in Schritt 8 soeben die Programmzeile 4 abgearbeitet haben (die wir – weil es ja eine Schleife ist – bereits zum dritten Mal durchlaufen haben) und die Arraywerte unverändert sind, die Variable **`sum`** den Wert 12 und die (Schleifen-)Variable **`i`** den Wert 2 enthält. Die Variable **`mw`** hat noch keinen Speicherplatz zugewiesen bekommen.

Außerdem können wir Angaben über das Laufzeitverhalten dieser Methode machen: Wenn die CPU unseres Computers für jede Zeile eine Millisekunde Bearbeitungszeit bräuchte, so würden die Methode (bei dem Eingabearray mit drei Elementen) also 13 Millisekunden benötigen. Offensichtlich würde ein Array mit vier Elementen zwei Millisekunden länger (also 15 Millisekunden) benötigen (**Erkläre:** warum?).

Übung: Gib den Zeitbedarf an, wenn das Array die Größe **a)** 6 **b)** n hätte!

Gültigkeitsbereich einer Variablen („scope“)

In der Beispieltabelle sieht man auch, dass Variablen ihren reservierten Speicherplatz wieder verlieren können. Der Speicher von Variable **`i`** wird nach Abarbeitung der Schleife wieder freigegeben (Siehe Schritt 11).

Ein Sonderfall ergibt sich bei Programmabschnitten folgender Art:

```
1   int i = 5;
2   if( i < 10) {
3      int i;
4      i = 8;
5   }
6   i = i + 1;
```

Normalerweise ist es verboten, Variablen „doppelt“ zu deklarieren (siehe Zeile 1 und 3) aber die Zeilen 3 und 4 sind mit „{„ und „}“ eingeklammert und bilden einen eigenen Block bzw. „scope“. In Blöcken darf man Variablennamen erneut benutzen („lokale Variable“). Dabei geht der Wert der „alten“ Variablen nicht verloren, sondern wird nur überdeckt. Für den Programmabschnitt erhalten wir:

Schritt	Prog.Zeile	i	i (lokal)
1	1	5	----
2	2		----
3	3	(verdeckt)	?
4	4	(verdeckt)	8
5	6	6	----

Zugegebenermaßen ist der Programmabschnitt völlig sinnlos, aber er zeigt, wie „lokale Variablen“ funktionieren.