Programmierkurs

Vorlesung 03

M.Sc. Laslo Hunhold

Department Mathematik/Informatik
Abteilung Informatik
Universität zu Köln

23. November 2020

Letzte Vorlesung

- Literale/Konstanten
- Vertiefung Typumwandlung
- Operationen auf Variablen
- Vertiefung Kontrollstrukturen

Vertiefung Kontrollstrukturen (Fortsetzung)

Steuerbefehl „`break`“

- Befehl zum Ausstieg (schon bei `switch` bekannt)
- Kann auch bei Schleifen (`while`, `for`, `do...while`) verwendet werden
- Steigt nur aus der *aktuellen* Schleife aus (bei Verschachtelungen)

```
1  int n;
2
3  for (n = 0; n <= 4; n++) {
4          if (n == 2) {
5                  break;
6          }
7          System.out.println(n);
8  }
```

```
0
1
```

Steuerbefehl „`continue`“

- Kann bei Schleifen (`while`, `for`, `do...while`) verwendet werden
- Springt zur Bedingungsprüfung der Schleife
- Bei `for` wird vorher noch der `innenausdruck` ausgeführt

```
1  int n;
2
3  for (n = 0; n <= 4; n++) {
4          if (n == 2) {
5                  continue;
6          }
7          System.out.println(n);
8  }
```

```
0
1
3
4
```

while versus do...while

```
int n = 0;
while (n > 0) {
        System.out.println("Joe");
}
```

```
int n = 0;
do {
        System.out.println("Joe");
} while (n > 0);
```

```
Joe
```

Operationen auf Variablen (Fortsetzung)

Assoziativität

- Betrachte „`a + b + c`“. Wie wird ausgewertet?
- Möglichkeit 1: Wie „`(a + b) + c`“ (Linksassoziativ)
- Möglichkeit 2: Wie „`a + (b + c)`“ (Rechtsassoziativ)
- Verhalten mit Klammern („Gliederungszeichen“) erzwingbar

„Addition ist mathematisch assoziativ, es macht keinen Unterschied.“
Achtung: Nebeneffekte, die in der Mathematik nicht vorkommen!

Zuweisungen im Ausdruck

```
int c = 4;
c = c + c + (c = 2);
/* c = 10, nicht 6 */
```

Auslöschung bei Ganzzahldivision

```
double z;
z = 100 / 50 / 10;   /* z = 0.0  */
z = (100 / 50) / 10; /* z = 0.0  */
z = 100 / (50 / 10); /* z = 20.0 */
```

Rang/Präzedenz

Gruppe	Operatoren	Assoziativität
1	`()` (Methodenaufruf), `[]` (später), `.` (später)	links
2	`++`/`--` (postfix)	
3	`++`/`--` (prefix), `+`/`-` (unär), `~`, `!`, `(TYP)`, `new` (später)	rechts
4	`*`, `/`, `%`	links
5	`+`, `-`	
6	`<<`, `>>>`, `>>`	
7	`<`, `<=`, `>`, `>=`, `instanceof` (später)	
8	`==`, `!=`	
9	`&`	
10	`^`	
11	`|`	
12	`&&`	
13	`||`	
14	`?:`	rechts
15	`=`, `OPERATOR=`	

- Je höher die Gruppenzahl, desto niedriger der Rang („Präzedenz“)
- Klammern werden implizit nach Präzedenz gesetzt:
  „`a + b / c`“ → „`a + (b / c)`“
  „`!a && b == true`“ → „`(!a) && (b == true)`“
- Bei Operatoren in derselben Gruppe nur nach Assoziativität:
  „`a + b - c`“ → „`(a + b) - c`“

Wahrheitstabelle Boolescher Operatoren

- Java hat keinen dedizierten logischen XOR-Operator
- Bitweise Operatoren `&`, `|` und `^` können auf `boolean` operieren; aber *nicht* empfohlen (Java-spezifische Funktionalität, nicht in C/C++/etc.)!
- XOR-Operator läßt sich allgemein (auch in C/C++/etc.) mit `!=` „emulieren“
- Typen beachten (keine implizite Umwandlung in `boolean`)!

`a`	`b`	`!a` (NOT)	`a && b` (AND)	`a || b` (OR)	`a != b` (XOR)
`false`	`false`	`true`	`false`	`false`	`false`
`true`	`false`	`false`	`false`	`true`	`true`
`false`	`true`	`true`	`false`	`true`	`true`
`true`	`true`	`false`	`true`	`true`	`false`

Unvollständige Auswertung bei && und ||

```
1  int n = 0;
2
3  if (false && (n = 1) > 0) {
4          ...
5  }
6  /* n = 0, nicht 1 */
```

```
1  int n = 0;
2
3  if (true || (n = 1) > 0) {
4          ...
5  }
6  /* n = 0, nicht 1 */
```

- Die Auswertung der BOOLEschen Operatoren && und || wird abgebrochen, sobald der Ausdruck garantiert `false` oder `true` ist.
- Passiert nicht mit & und |, aber Empfehlung: Bitweise Operatoren *nicht* als logische Operatoren verwenden!

Arrays

Motivation

- Nutzen Computer zur Datenverarbeitung
- Beispiel Messreihe; wie speichern wir die Daten?

```
double messreihe_wert1, messreihe_wert2, messreihe_wert3;
```

- Was wenn die Messreihe aus 1000 Messungen besteht?
- Wollen viele Einzelwerte in einer „Reihe“ anordnen

Lösung: Arrays

- Bitgruppen werden im Speicher in einer Reihe angeordnet
- Beispiel: Array aus vier int8:

```
01011101 01100101 10011011 10011011
6        7        8        9
↑
Startadresse
```

Deklaration

- Typname wird mit `[]` versehen (Konvention)

```
int[] arr;
```

- Alternativ: Variablenname wird mit `[]` versehen (C-Stil)

```
int arr[];
```

- Deklariert Variable mit der Bezeichnung `arr` vom Typ `int`-Array
- Es wird keine Elementanzahl vorgegeben
- Ein Array ist ein komplexer Datentyp (da variable Länge)
- Der Array existiert (noch) nicht; `arr` ist nur eine Variable, die eine Speicheradresse (die Startadresse) enthalten kann (`arr` ist ein „Zeiger“)

Initialisierung

`new`-Operator

- Erzeugt eine Instanz des angegebenen komplexen Typs

```
int[] arr;
arr = new int[5];
```

- Länge muß angegeben werden, Array ist noch nicht mit Daten gefüllt
- Alternativ: Explizite Angabe der Einträge dahinter

```
arr = new int[]{1, 2, 3, 4, 5};
```

Arrayliteral

- Explizite Angabe (nur bei der Deklaration erlaubt!)

```
double[] arr = { 1.0, 2.2, 3.3, 4.1, 5.4 };
```

Nullzeiger

- Literal `null` Speicheradresse ins „Nirgendwo“ (meistens die Adresse 0, siehe `NULL` in C/C++)
- Initialisierung/Prüfung von Zeigern, die nirgendwohin zeigen

```
int[] arr = null;
```

Zugriff und Information

Arrayzugriff

- Operator `[]`
- Zugriff auf ein Element eines Arrays
- Zwei Operanden: Array-Variable und ganzzahliger „Offset“ (beginnend bei 0)
- z.B. `arr[0]` gibt das 1. Element (Offset = 0) des Arrays `arr` zurück

Arraylänge

- Ein Array `arr` hat die Länge `arr.length`
- Achtung! Das letzte Element ist `arr[arr.length - 1]` und nicht `arr[arr.length]`

Beispiele

Schleife über Array, alle Elemente auf Null setzen

```
1  int[] arr = new int[20];
2
3  for (int i = 0; i < arr.length; i++) {
4          arr[i] = 0;
5  }
```

Verwendung von `null`

```
1  int n = 5;
2  int[] arr = null;
3
4  if (n < 0) {
5          arr = new int[10];
6  }
7  if (arr == null) {
8          arr = new int[n];
9  }
10 System.out.println(arr.length);
```

Lokale Variablen

Ausblick Variablenarten

- Lokale Variablen (Innerhalb von Blöcken/Methoden)
- Instanzvariablen (später)
- Statische Variablen (Klassenvariablen) (später)

Gültigkeitsbereich und Lebensdauer

- Lokale Variable hat von dort an, wo sie definiert wird, bis zum Ende des Blocks/der Methode „Gültigkeit“
- Diesen Bereich nennt man „Gültigkeitsbereich“ („scope“)
- Nur innerhalb ihres Gültigkeitsbereichs kann auf eine Variable zugegriffen werden
- Bei Variablen mit gleichem Namen dürfen sich die Gültigkeitsbereiche nicht überschneiden

```
for (int i = 0; i <= 10; i++) {
        ...
}
for (int i = 0; i <= 15; i++) {
        ...
}
```

```
int i = 10;
for (int i = 0; i <= 10; i++) { /* Fehler */
        ...
}
```

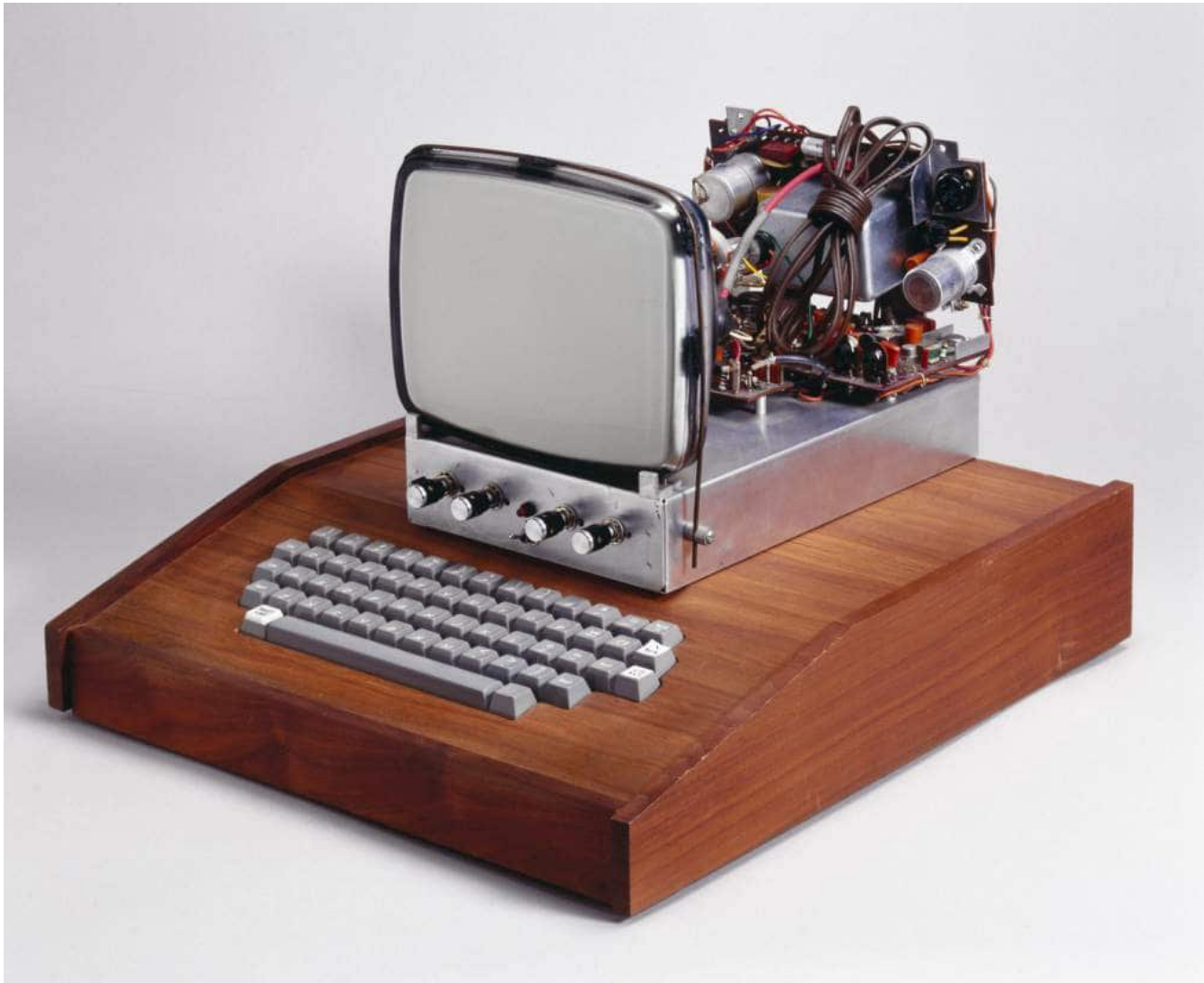

Abbildung: Apple I (1976) (Quelle: SSPL/Getty)