Programmierkurs

Vorlesung 05

M.Sc. Laslo Hunhold

Department Mathematik/Informatik
Abteilung Informatik
Universität zu Köln

7. Dezember 2020

Letzte Vorlesung

- Mehrdimensionale Arrays
- Klassen
- Programmstruktur (Packages, Klassen, Namensräume)
- Instanz („Objekt“) einer Klasse
- Instanz- versus Klassenvariablen

Standardwerte für Typen

- `byte`: 0
- `short`: 0
- `int`: 0
- `long`: 0L
- `float`: 0.0f
- `double`: 0.0d
- `boolean`: false
- `char`: '\0' (Nullzeichen, später)
- Komplex (i.e. Zeiger) (Array, Klasse, ...): `null`

Achtung! Lokale Variablen werden bei Deklaration *nicht* auf Standardwerte gesetzt

Vertiefung zu Objekten

Klasse

- Klasse ist ein Bauplan

```
class Bruch {
        int zaehler;
        int nenner;
}
```

- Zeiger speichert nur Adresse (z.B. als uint64 auf 64-Bit-Systemen)

```
Bruch b;
```

- Objekt wird nach Plan „gebaut“ (i.e. Speicher reserviert) und Adresse im Zeiger `b` gespeichert

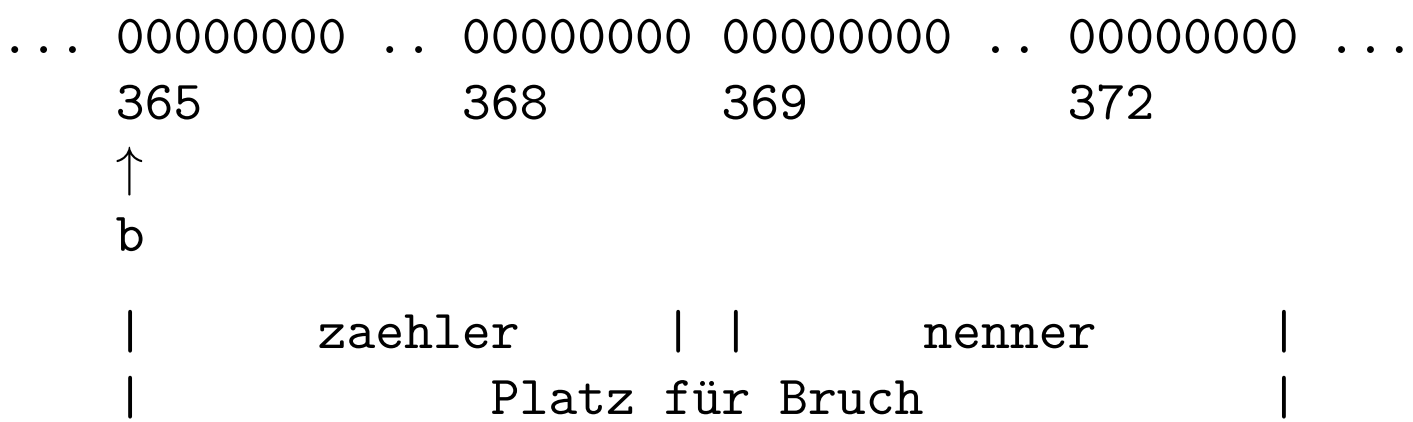
```
b = new Bruch();
```

- Instanzvariablen werden auf Standardwerte gesetzt (Klassenvariablen („`static`“) auch, bei Programmstart)

Speicherstruktur

```
class Bruch {
        int zaehler;
        int nenner;
}
```

```
Bruch b = new Bruch();
/* z.B. b = 365 */
```



Primitiver versus Komplexer Datentyp

```
int a = 4;
int b = a;
```

- Zwei Variablen `a` und `b` (Integer)
- Der Wert von `a` wird in `b` kopiert

```
Bruch a = new Bruch();
Bruch b = a;
```

- Zwei Variablen `a` und `b` (Adressen)
- Nur ein Objekt
- Es wird nur die Adresse von `a` in `b` kopiert
- Zeiger `b` ist ein „Alias“ für das Objekt, auf das `a` zeigt

Lebensdauer von Objekten

```
Bruch b = new Bruch();
```

- Lokale Variable `b` ist nur ein Zeiger; verliert Gültigkeit am Blockende
- Was passiert dann mit dem Objekt?
- Objekt wird automatisch verworfen („deallokiert“), wenn kein Zeiger mehr existiert, der auf das Objekt zeigt.

Vertiefung zu mehrdimensionalen Arrays

Speicherstruktur

```
int[][] arr = {
        { 10, 20, 30, 40, 50 },
        { 3, 6, 12, 32, 64 },
        { 7, 11, 22, 33, 44 },
};
```

- `arr` ist vom Typ `int[][]`, also ein Zeiger auf einen `int[]`-Array (Array von Zeigern)
- `arr` $\rightarrow$ `{ arr[0], arr[1], arr[2] }`
- `arr[0]`, `arr[1]` und `arr[2]` sind vom Typ `int[]`, also Zeiger auf `int`-Arrays
- `arr[0]` $\rightarrow$ `{ 10, 20, 30, 40, 50 }`
- `arr[1]` $\rightarrow$ `{ 3, 6, 12, 32, 64 }`
- `arr[2]` $\rightarrow$ `{ 7, 11, 22, 33, 44 }`

```
int[][] arr = { null, null, null };
```

Adressierung

```
int[][] arr = {
        { 10, 20, 30, 40, 50 },
        { 3, 6, 12, 32, 64 },
        { 7, 11, 22, 33, 44 },
};
```

Wie erreichen wir die 32?

- 2. Zeile, also Array, auf den `arr[1]` zeigt
- 4. Spalte, also Offset 3
- Ergebnis: `arr[1][3]`

Zeichen und Zeichenketten

Datentyp `char`

- Typ uint16; jeder Zahl ist ein sogenannter „Codepoint“ zugewiesen
- Historisch: Viele verschiedene inkompatible Zuweisungssysteme (US-ASCII, ISO/IEC 8859-*, etc.)
- Heute: Unicode (`https://unicode.org`)
- Beispiel: Zahl 65 (0x41) steht für den Buchstaben „`A`“ und 63 (0x3F) steht für das Zeichen „`?`“ (Schreibweise U+0041 und U+003F)
- Achtung! Ein Codepoint ist nicht immer ein Zeichen (Stichwort: Grapheme Clusters)
- Komplexe Details; „Codepoint = Zeichen“ kann erstmal angenommen werden und gilt für die „einfachen“ Zeichen

Literale

- Buchstabe umfaßt mit einfachen Anführungszeichen (`'A'`, `'\u0041'`) oder Wert (65, `0x41`)
- Steuer-/Sonderzeichen: `'\t'` (Tab), `'\n'` (Zeilenumbruch), `'\0'` (Nullzeichen), etc.
- Literales einfaches Anführungszeichen: Nicht `'''`, sondern `'\''`
- Literales Backslash: Nicht `'\'`, sondern `'\\'`

Datentyp `String`

- Komplexer Datentyp (Klasse), vereinfacht

```
class String {
        char[] s;
}
```

- Speichert Zeichenketten

Literale

- Zeichenkette umfaßt mit doppelten Anführungszeichen (z.B. `"Hallo Welt\n"`)
- Literales doppeltes Anführungszeichen: Nicht `"..".."`, sondern `"..\".."`
- Literales Backslash im String: Nicht `"..\.."`, sondern `"..\\.."`

Stringwerkzeuge

Warum nicht statt `String` einfach einen `char[]` benutzen?

```
1  String s = "Hallo";
2
3  System.out.println(s.length());
4  System.out.println(s.isEmpty());
5  System.out.println(s.indexOf('H'));
6  System.out.println(s.charAt(1));
7  System.out.println(s.concat(" Welt!"));
8  System.out.println(s + " Welt!");
```

```
5
false
0
a
Hallo Welt!
Hallo Welt!
```

Beispiele

Implizite Typumwandlung bei Konkatenation

```
System.out.println("Hallo" + " Welt " + 2019 + "!");
```

```
Hallo Welt 2019!
```

Steuer-/Sonderzeichen

```
System.out.println("\tHallo\n\t\tWelt");
```

```
        Hallo
                Welt
```

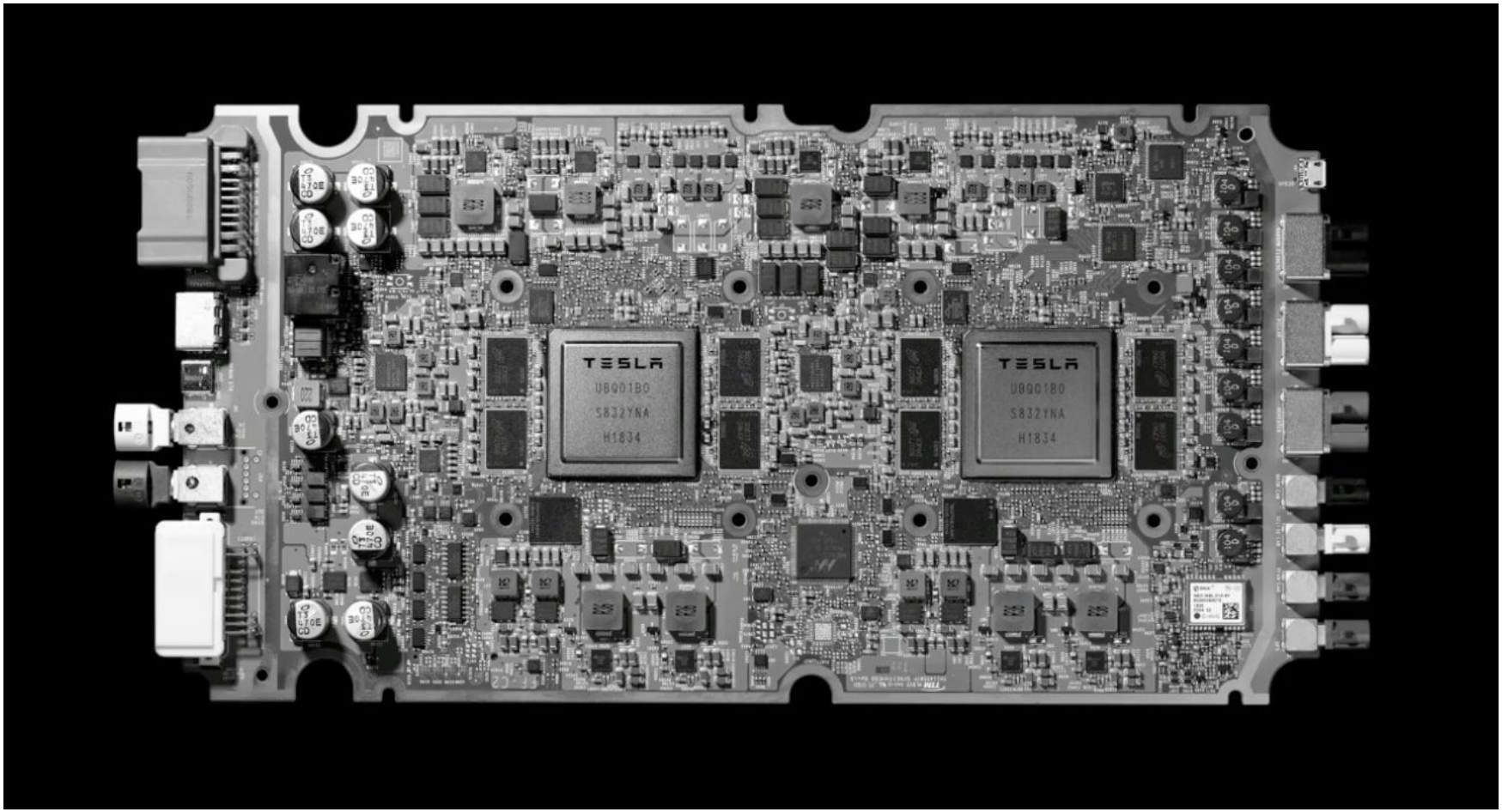


Abbildung: Teslas „FSD Computer“ (2019) („Full Self-Driving“, doppelt redundant) (Quelle: Tesla Inc., Autonomy Investor Day, 22. April 2019)