Informatik II

Woche 10, 09.03.2017

Giuseppe Accaputo

g@accaputo.ch

Nachbesprechung: **Übung 2**

Übung 2: Abgaben

- Habe von allen Abgaben erhalten!

Kompliment an euch alle!

Tests laufen lassen vor Submit

- **Wichtig:** `@RunTests` auskommentieren (`//` entfernen) bevor ihr submitted:

```
@RunTests
public class Main { … }
```

Ganzzahldivision

Regeln:

- `int / int = int`
- `int / double = double`
- `double / int = double`

Assoziativität

Faustregel:

1. Punkt vor Strich
2. Arithmetisch vor Vergleich
3. Vergleich vor Logisch
4. ! vor && vor ||

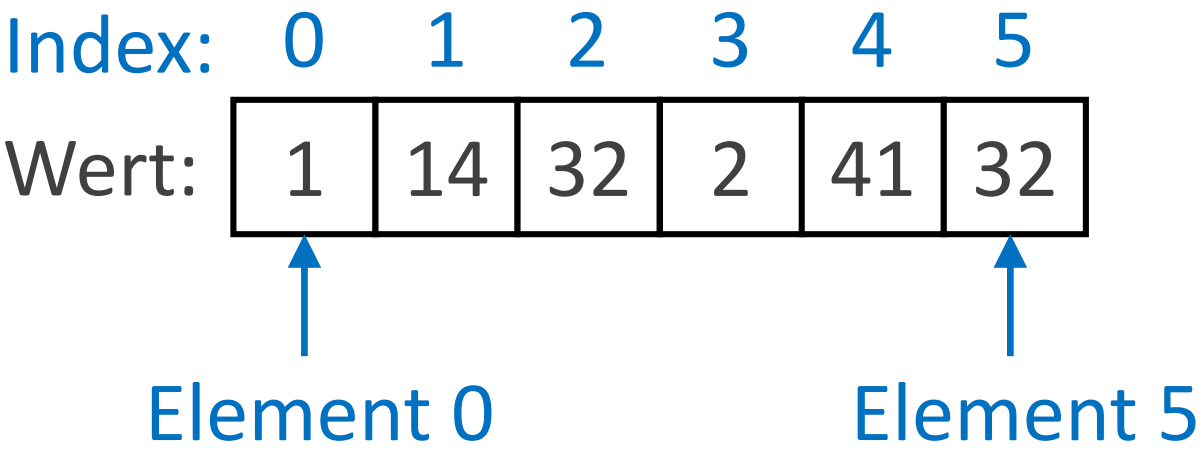
Musterlösung Übung 2

- Blick in die Musterlösung werfen nach Abgabe kann lehrreich sein
- Kurze Besprechung der Musterlösungen
- Elegante Implementation zu Aufgabe 2.4

Repetition:
Strings, Arrays, Pass by Value

Arrays

- **Array**: Objekt, welches mehrere Werte desselben Typs speichert
- **Element**: Ein Wert an einem Index des Arrays
- **Index**: Position eines Elementes im Array. Startet bei 0

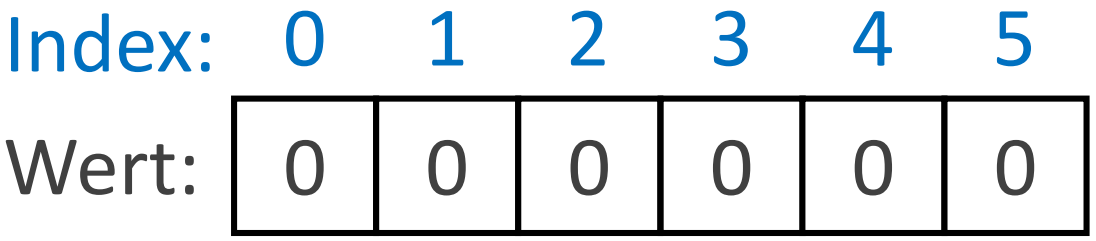


Arrays: Deklaration

```
Typ[] name = new Typ[Länge];
```

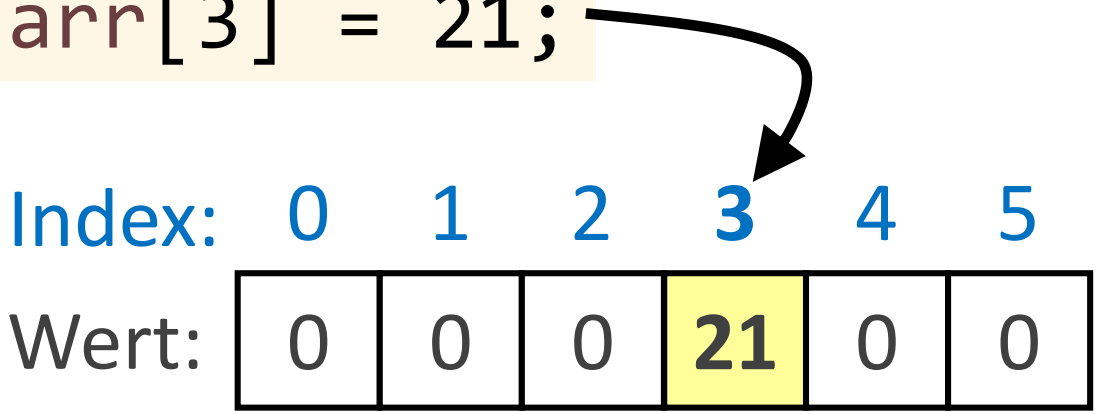
- Beispiel:

```
int[] arr = new int[6];
```



Arrays: Elementzugriff

- Element lesen: `int a = arr[index];`
- Element speichern: `arr[3] = 21;`



- Erste Position auf Index `0`, letzte auf `arr.length - 1`
- **Wichtig**: Exception wird geworfen bei Fehlzugriff!

Arrays: Durchiterieren

- Arrays besitzen die Instanzvariable `length`, welche die Länge des Arrays abspeichert
- Um durch ein Array zu iterieren, kann man eine `for`-Schleife und das `length` Attribut verwenden:

```
int[] arr = new int[3];
arr[0] = 1; arr[1] = 2; arr[2] = 3;

for(int i = 0; i < arr.length; i++)
    System.out.println(arr[i] + "");
```

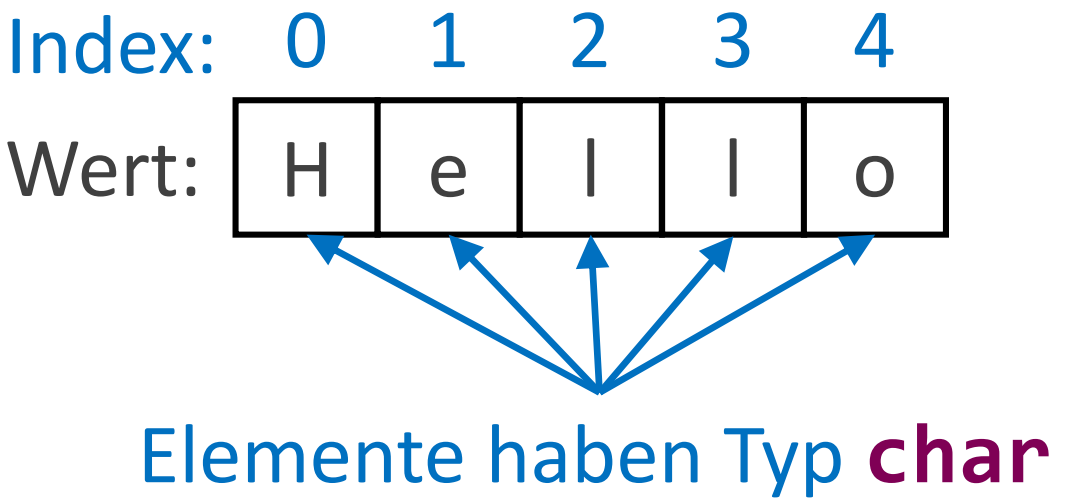
Konsole:

```
1 2 3
```

Strings

- String: Objekt, dass eine Zeichenkette speichert

```
String name = "Hello";
```



Strings

- String Objekte sind *immutable*, d.h. sie können nicht verändert werden:

```
String s = "Hello, World!";
s[1] = 'H'; // Kompilierfehler
```

String Operationen

- Konkatenation zweier Strings mittels +-Operator:

```
String s = "Hello";
String t = ", World!";
String u = t + s;
// u = "Hello, World!"
```

- `i`-ter Buchstabe eines Strings auslesen:

```
String s = "Hello";
char c = s.charAt(1); // c == 'e'
```

String Operationen

- Bestimmte Buchstaben ersetzen:

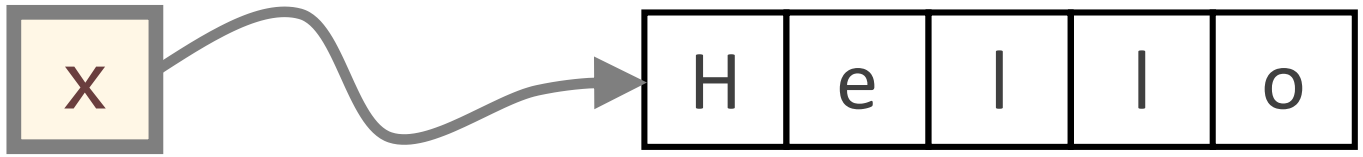
```
String s = "Hello";
String t = s.replace('e', 'a');
// t = "Hallo"
// s = "Hello" (nicht verändert)
```

  - **Wichtig**: Da Strings *immutable* sind, arbeitet `replace` auf einer Kopie des Strings `s`, welche wir in `t` dann zwischenspeichern. `replace` führt also kein in-place Replacement durch!

Strings: Zeichenketten vergleichen

- Variablen sind Referenzen auf Werte im Speicher

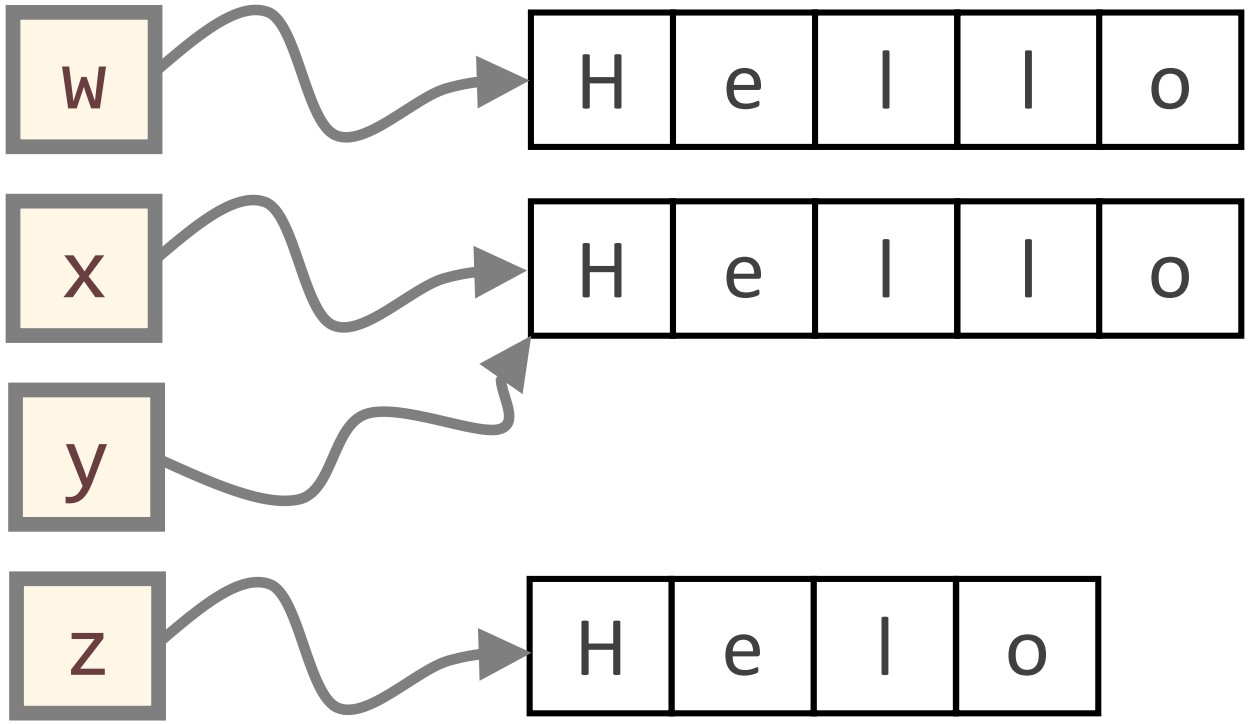
```
String x = "Hello";
```



- == und != Operatoren auf Strings machen einen *Referenzvergleich* und keinen Zeichenkettenvergleich
- `equals`-Methode verwenden um Zeichenketten zu vergleichen

Strings: Zeichenkettenvergleiche

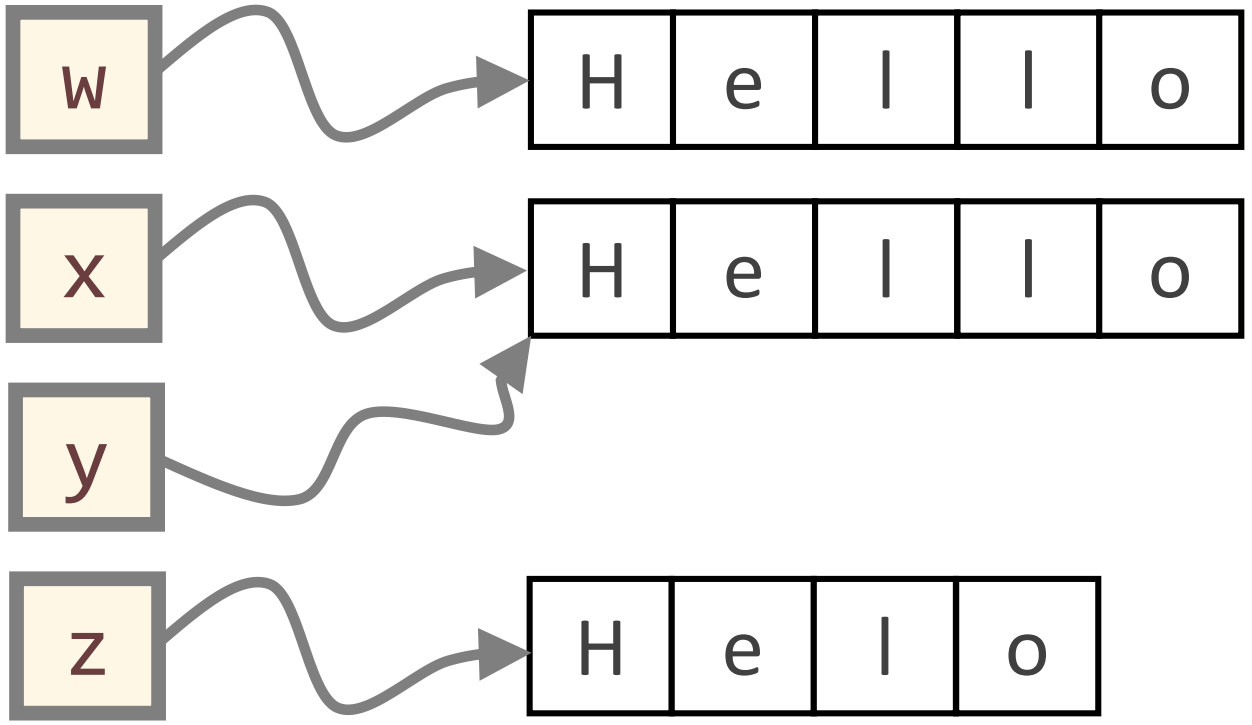
```
String w = "Hello";
String x = "Hello";
String y = x;
String z = "Helo";
```




```
w == w → true
w == x → false
x == y → true
```

Strings: Zeichenkettenvergleiche

```
String w = "Hello";
String x = "Hello";
String y = x;
String z = "Helo";
```




```
w.equals(x) → true
x.equals(y) → true
y.Equals(z) → false
```

Allgemein: Vergleiche

- Bei primitiven Datentypen (`boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`):
  == vergleicht Werte der beiden Variablen miteinander

```
char c1 = 'h';
char c2 = 'h';
boolean b = (c1 == c2); // true
```

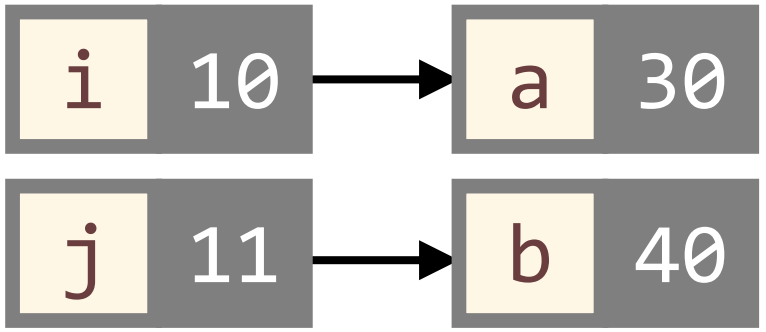
- Bei den restlichen Datentypen (`String`, `MyClass`, etc.):
  == vergleicht die Referenzen der Variablen miteinander

Java ist Pass-By-Value

- Parameter: Primitive Datentypen werden kopiert

```
void do(int a, int b){
    a = 30;
    b = 40;
}

int i = 10, j = 11;
do(i,j);
```



Nach Aufruf von do(i,j):

i == 10

j == 11

Java ist Pass-By-Value

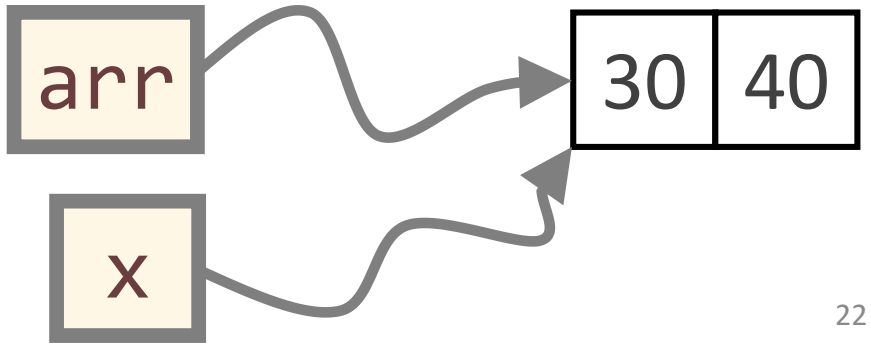
- Parameter: Referenzen auf Objekte werden kopiert

```
void do(int x[]){
    x[0] = 30;
    x[1] = 40;
}

int[] arr = new int[2];
arr[0] = 10; arr[1] = 11;
do(arr);
```

Zu Beginn:

Nach Aufruf von do(arr):



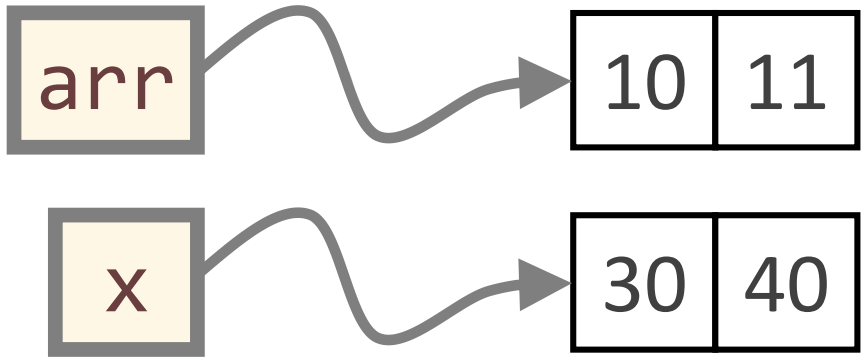
Java ist Pass-By-Value

```
void do(int x[]){
    x = new int[2];
    x[0] = 30;
    x[1] = 40;
}

int[] arr = new int[2];
arr[0] = 10; arr[1] = 11;
do(arr);
```

Zu Beginn:

Nach Aufruf von do(arr):



Aufgabe Pass-By-Value

- Welche Elemente befinden sich im Array x?

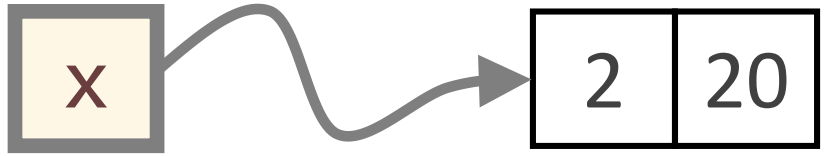
```
public static void d(int[] y){
   y[0] = 2;
   y = new int[2];
   y[1] = 3;
}
…

int[] x = new int[2];
x[0] = 100; x[1] = 200;
d(x);
```

Lösung Pass-By-Value

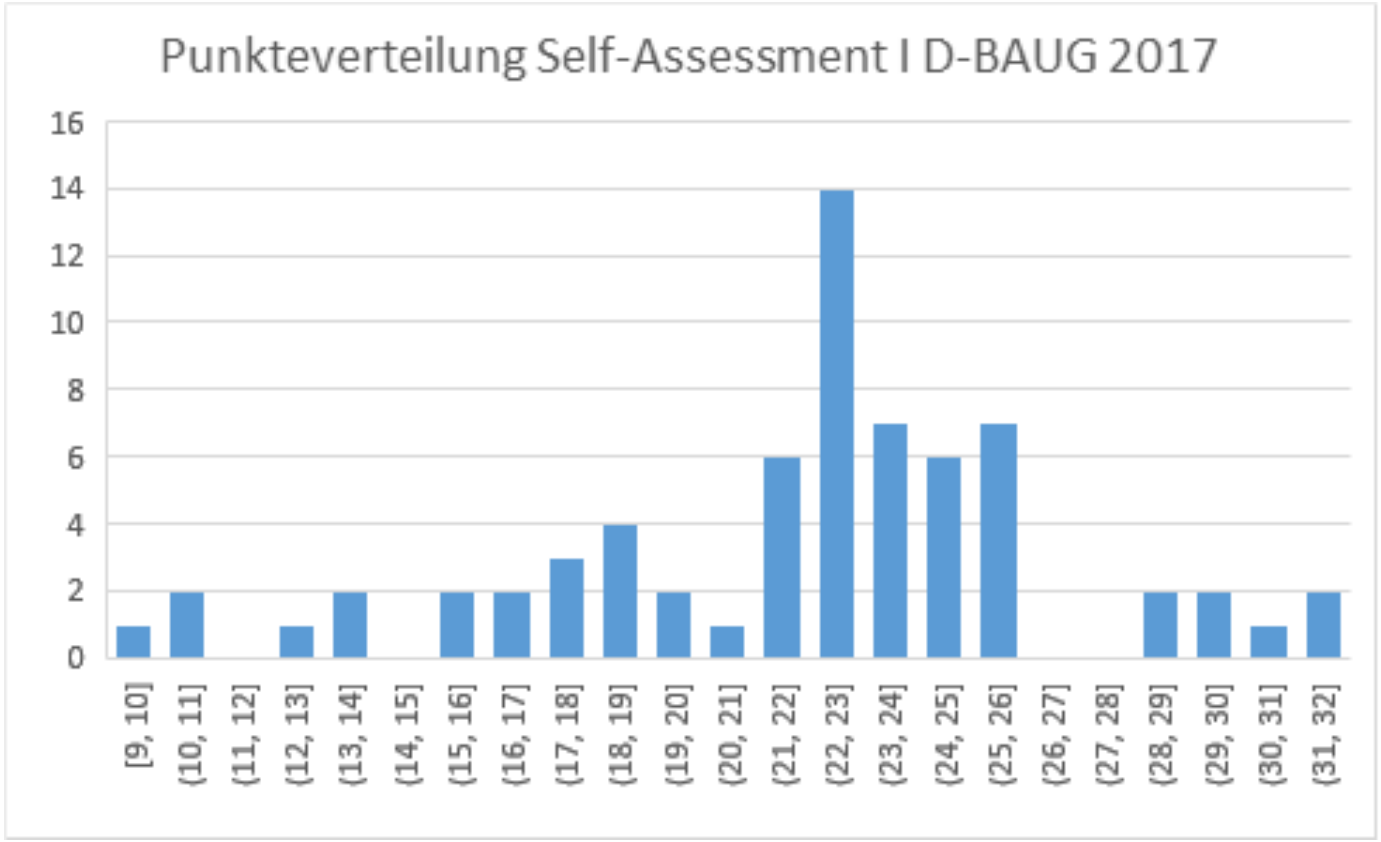
```
public static void d(int[] y){
   y[0] = 2;
   y = new int[2];
   y[1] = 3;
}
…

int[] x = new int[2];
x[0] = 10; x[1] = 20;
d(x);
```



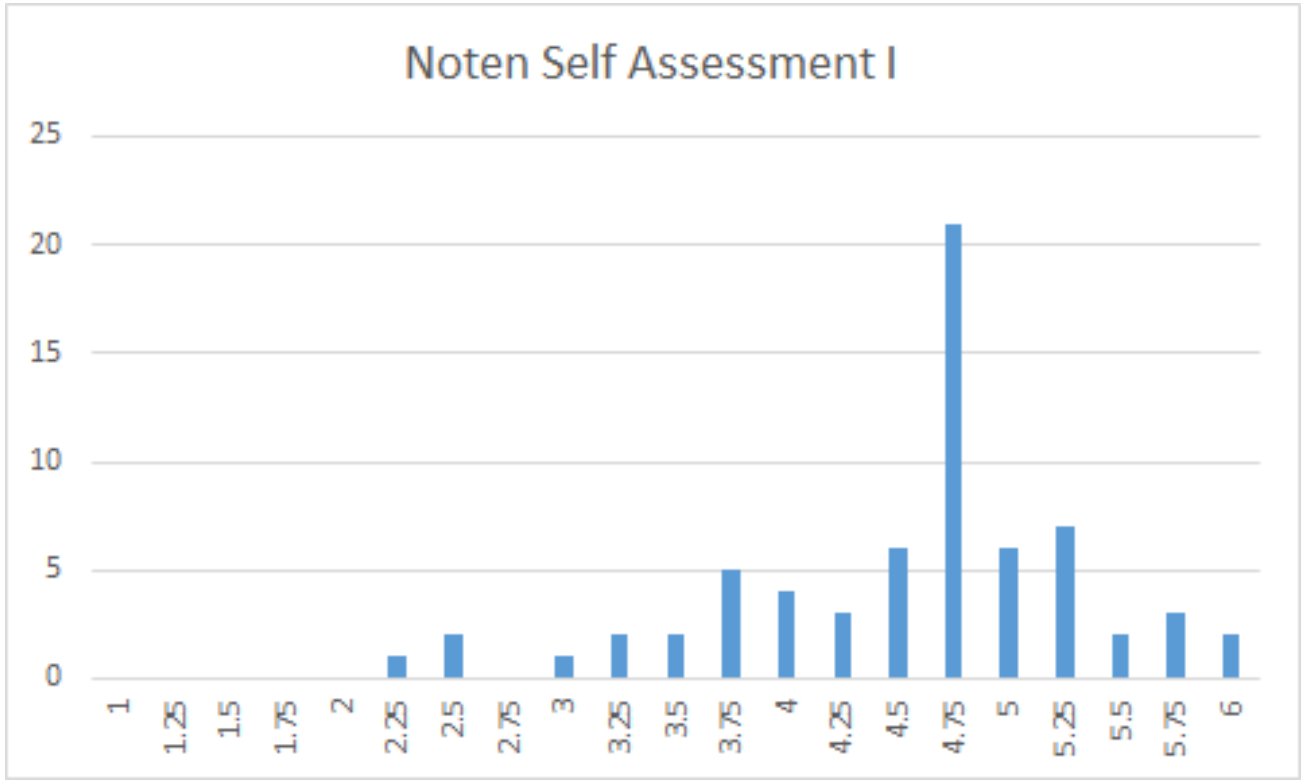
Self-Assesment Test

Self-Assessment Test: Punkteverteilung



Self-Assessment Test: Notenverteilung

- 19 Punkte = 4.0 | 32 Punkte = 6.0 | 3 Punkte = 1.0



Vorbesprechung: **Übung 3**

Fragen oder Anregungen?