

Wiederholung

Verzweigungen

Beispiel:

```
if ( x > 5 ) {  
    // Anweisungen  
    // ...  
}  
  
else if ( x > 10 ) {  
    // Anweisungen  
    // ...  
}  
  
else {  
    // Anweisungen  
    // ...  
}
```

- Es kann beliebig viele **else if**-Blöcke geben (auch gar keinen)
- Der **else**-Block ist optional

- Die UND-Verknüpfung ($\&\&$) von zwei Bedingungen ist wahr, wenn **jede** der Bedingungen wahr ist
- Die ODER-Verknüpfung ($\|\ \|$) von zwei Bedingungen ist wahr, wenn **mindestens eine** der Bedingungen wahr ist
- Die XOR-Verknüpfung (\wedge) von zwei Bedingungen ist wahr, wenn **genau eine** der Bedingungen wahr ist
- Die Negation (vorangestelltes $!$) einer Bedingung verkehrt ihren Wert ins Gegenteil

- Der Modulo-Operator % liefert den Divisionsrest einer Ganzzahldivision
- Verwendungsmöglichkeiten:
 - Einschränken einer Zählvariable auf einen bestimmten Bereich
 - Teilbarkeit untersuchen, um bspw. gerade von ungeraden Zahlen zu unterscheiden

Überladen von Methoden

- Die Methode `fill` (und einige andere) gibt es in verschiedenen Varianten, zum Beispiel:
 - `fill(128)` Graustufen
 - `fill(255, 0, 255)` RGB
- Unterschiedliche Methoden mit gleichem Namen sind eigentlich nicht erlaubt:

```
void machWas(int farbe) {  
    // Anweisungen  
    // ...  
}  
  
void machWas(int groesse) {  
    // Anweisungen  
    // ...  
}
```

Geht nicht!

- Wieso klappt das bei `fill()`?

- Alle Versionen von `fill()` haben eine unterschiedliche **Signatur**
- Signatur einer Methode:
 - Methodenname und Datentypen aller Parameter
- Wichtig:
 - Rückgabetyt und Parameternamen sind **nicht** Teil der Signatur!

Signatur einer Methode - Beispiele

Methode	Signatur
<code>void machWas(int x, double y)</code>	<code>machWas(int, double)</code>
<code>byte machWas(int x, double y)</code>	<code>machWas(int, double)</code>
<code>boolean istOk(int w)</code>	<code>istOk(int)</code>
<code>double quadriere(double x)</code>	<code>quadriere(double)</code>

- Es darf genau dann mehrere Methoden mit gleichem Namen geben, wenn sie alle eine unterschiedliche Signatur haben
- In diesem Fall spricht man vom **Überladen** einer Methode bzw. einer **überladenen** Methode

Aufgabe 1

- Schreibe drei Methoden mit dem Namen **dividiere**, die über die zwei Parameter **divident** und **divisor** verfügt und daraus den zugehörigen Quotienten berechnet und als Ergebnis zurückliefert
 - Die erste Methode soll mit Parametern vom Typ **int** arbeiten und einen Wert vom Typ **int** zurückgeben
 - Die zweite Methode soll mit Parametern vom Typ **float** arbeiten und einen Wert vom Typ **float** zurückgeben
 - Die dritte Methode soll mit Parametern vom Typ **double** arbeiten und einen Wert vom Typ **double** zurückgeben
- Zusatzaufgabe:
 - Was passiert, wenn man den Ergebnistyp der ersten Methode von **int** auf **double** ändert?
 - Woher “weiß” Processing, ob man die Methode 2 (**float**) oder Methode 3 (**double**) aufrufen möchte?

- Unterschied zwischen Processing und “richtigem” Java:
Datentyp einer Konstante wie 5.0:
 - In Processing: `float`
 - In Java: `double`
- Der Datentyp kann auch explizit angegeben werden:
 - `5.0d`: `double`
 - `5.0f`: `float`