

# Imperative Programmierung

---

Wie “sagen” wir einem Computer, was er tun soll?

Wir schreiben ein Programm!

Der Haken bei der Sache: Wir müssen eine Sprache wählen, die der Computer versteht!

Hier im Kurs: **Java**

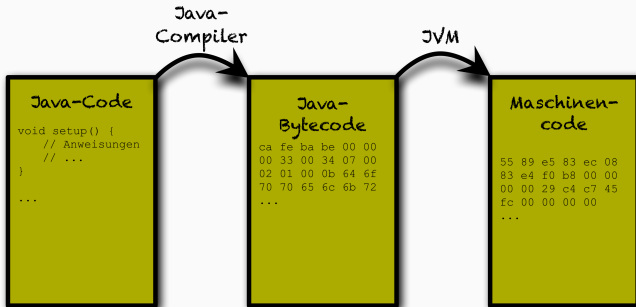


Abbildung 1: Übersetzung eines Programms

- Programmbibliothek und (einfache) integrierte Entwicklungsumgebung (IDE)
- Programmiersprache: Vereinfachtes Java
- Spezialisiert auf den Bereich Grafik

## Ein (sehr) einfaches Programm

```
1  void setup() {  
2      size(800, 600);  
3      circle(400, 300, 100);  
4  }
```

Zeile 1 Beginn der Definition der Methode `setup()`

Zeile 2 Aufruf der Methode `size()` zum Setzen der Fenstergröße

Zeile 3 Aufruf der Methode `circle()` zum Zeichnen eines Kreises

Zeile 4 Ende der Definition der Methode `setup()`

# Aufgabe 1

- Melde Dich am Computer an und starte Processing
- Übertrage das Beispielprogramm und führe es aus
- Ergänze das Programm, sodass zwei weitere Kreise an andere Stellen gezeichnet werden.
- Rufe die Methode `stroke()` auf, um die Linienfarbe der Kreise zu ändern
  - die Methode erwartet in Klammern 3 Werte zwischen 0 und 255 (rot, grün und blau)

```
void setup() {  
    size(800, 600);  
    circle(400, 300, 100);  
}
```

## Methoden Teil 1

---

- Eine Methode ist eine Art Unterprogramm mit eigenem Namen
- Man kann eigene Methoden *definieren* und vorhandene Methoden *aufrufen*
- Nach der “Abarbeitung” einer aufgerufenen Methode wird die Codeausführung nach dem Methodenaufruf fortgesetzt



# Methoden ohne Parameter

Definition:

```
void machWas() {  
    // Anweisungen  
    // ...  
}
```

Aufruf (z. B. in `setup()`):

```
machWas();
```

**void** gibt an, dass die Methode kein Ergebnis zurückgibt

**machWas** ist der (innerhalb gewisser Regeln) frei wählbare Name der Methode

**()** gibt an, dass die Methode keine Parameter erwartet

**{** leitet die eigentliche Definition der Methode ein

**}** beendet die Definition der Methode

**//** Leitet einen Kommentar ein (wird vom Compiler ignoriert)

- Schreibe eine Methode `void doppelKreis()`, die zwei überlappende Kreise (ungefähr) in die Mitte des Zeichenfensters zeichnet
- Rufe diese Methode innerhalb der Methode `setup()` auf.
- Hinweis: Die Methode `noFill()` sorgt dafür, dass nur Umrisse ohne Füllung gezeichnet werden.

- Die Methode `doppelKreis()` ist unflexibel: Die Kreise werden immer auf die gleiche Art und Weise und an die gleiche Stelle gezeichnet.
- Wünschenswert: Beim Methodenaufruf festlegen, wohin die Kreise gezeichnet werden sollen.
- Lösung: Methode mit *Parametern*

Definition:

```
void doppelKreis(int x, int y) {  
    circle(x - 25, y, 100);  
    circle(x + 25, y, 100);  
}
```

Möglicher Aufruf:

```
doppelKreis(400, 300);
```

**int x** Legt fest, dass der erste Parameter der Methode den Namen **x** (frei wählbar) und den Datentyp **int** (ganze Zahl) hat.

**int y** Legt fest, dass der zweite Parameter der Methode den Namen **y** und den Datentyp **int** hat.

Eine Methode kann beliebig viele Parameter haben.

- Erweitere die vorhandene Methode `doppelKreis()` um drei Parameter für die x- und y-Koordinate sowie den Radius der beiden Kreise
- Rufe diese Methode innerhalb von `setup()` mehrfach mit unterschiedlichen Werten für die Parameter auf

## Aufgabe 4

- Schreibe eine Methode `nikoHaus()`, die das Haus vom Nikolaus zeichnet
  - Die Position der linken, unteren Ecke soll mittels geeigneter Parameter angegeben werden können
  - Verwende die vorhandene Methode `line(int x1, int y1, int x2, int y2)`, die eine Linie vom Punkt (x1|y1) zum Punkt (x2|y2) zeichnet.
- Zusatz:
  - Erweitere die Methode so, dass auch die Größe des Hauses durch einen Parameter festgelegt werden kann
- Rufe die Methode wieder mehrfach mit unterschiedlichen Werten für die Parameter auf.