# Java für Beginner > #03 OOP Cheat Sheet

by [deleted] via cheatography.com/69240/cs/17462/

## Einleitung

Java hat einen ziemlich "OOP-Lastigen" Programmierstyl (Objekt-Orientierte Programmierung). Das heißt das jedes Objekt seine eigene Identität. Ein Apfel ist Ein Objekt, welches anders ist.

In der OOP hat jedes Objekt seine eigene Identität. Ein Apfel ist Ein Objekt, welches anders ist. Auch 2 Äpfel sind 2 Objekte, welche sich unterscheiden. Auch 2 Tasse. Auch 2 Äpf

**Klassen** beschreiben wie ein Objekt sein wird / was es haben wird.

In anderen Worten können **Klassen wie BluePrints abgesehen werden.**

Man kann die selbe Klasse als BluePrint für unendlich Objekte benutzen.

Jede Klasse hat einen Namen, Variablen (Attribute) und Methoden (Verhalten). Hier ist ein Beispiels-Bild:

https://api.sololearn.com/DownloadFile?id=2429

## Funktionen & Variablen

**Methoden (Auch Funktionen/Verhalten gennant)** sind Sammlungen von Statements um eine Operation durchzuführen. Wie zB. auch die Methode `println(x)` in `System.out`.

Funktionen die ein `static`-Keyword besitzen **können** auch nur mit Hilfe der Klasse (BluePrints) aufgerufen werden, aber dann haben diese Methoden auch nur die `static`-Variablen/Funktionen der Klasse zu Verfügung.

```
class MyClass {

  public int var = 2;

 void hallo(){ System.ou t.p rin tln ("Ha llo "); }
 int addier en(int x1, int x2){ return x1 + x2; }

  static void test(){
    thi s.var= 2;// >Fe hler:  Die Variable " var " kann nur eine erstellte Instanz besi
tzen.
 }

}
```

**Variablen**

Eine Klasse kann **"2 Typen von Variablen"** haben:

- Variable mit `static`-Keyword, welche für das gesamte BluePrint gilt

- Variable ohne `static`-Keyword, welche für jedes Objekt einzeln gilt.

Im ersten Fall ist die Variable `var` eine Klassenvariable.

Im zweiten Fall ist die Variable `var` eine Objektvariable.

## Funktionen & Variablen (cont)

**Objekte können miteinander interagieren.**

Siehe **Konstruktor** um zu sehen wie man ein neues Objekt erstellt.

Siehe **Getter & Setter** für extrigen "Variablen-Schutz".

## Zugriff Modifizierer

| | |
|---|---|
| public | Für Jeden Zugreifbar |
| private | Nur für die Klasse |
| protected | Nur für die Klasse und Sub-Klassen |
| static | Keyword, damit man eine Methode/Variable auch nur durch die Klasse/das BluePrint bekommen kann |
| final | Keyword, dass eine Variable nicht veränderbar ist |

## Konstruktor

Eine Klasse kann außer Funktionen und Methoden auch sogenannte Konstruktöre haben.

Ein Konstruktor ist aufgebaut wie eine Methode, aber nur ist er dafür **ein neues Objekt seines Klasse zu erstellen.**

Ein Konstruktor sieht wie folgt aus: (Syntax)

```
public Klasse nName(...){ Code }
```

Ein Beispiel für die Klasse MyClass, welche eine Variable `var` hat. (aus "Getters und Setters"):

```
public MyClas s(int wert) { // Beim erstellen dieser Klasse muss ein We
  thi s.var = wert; // Setzt die Klasse n-V ariable " val " auf den mit
}
```

Und nun kann man mit:

```
MyClass instanz1 = new MyClas s(224);
```

Eine neue Instanz/Objekt der Klasse `MyClass` erstellen.

## Getter & Setter

**Getter & Setter** ist ein Konzept in der Programmierung, wo dafür gedacht ist dass man **das Setzen einer Variable überprüfen kann**, um Fehler zu vermeiden.

## Getter & Setter (cont)

Dies ist dann, wenn man Objekte sehr einfach zum "Kriegen" erstellt.

Ein klassisches Beispiel wäre es, dass man ei

```
class MyClass {

  private int var = 2;

 void setVar(int wert){
    if(wert < 0) { wert = 0; } //
    this.var = wert;
 }

  int getVar(){ return this.var; }

}
```

Hiermit kann ich mir ziemlich sicher sein, dass

In jedem Java Programm wird diese Programm sollte.

Kurz gesagt sind Getter und Setter eine "-Idioten-Schutz", damit Variablen keinen nicht vorhergesehen Wert erhalten!

## Wert VS Referenz

Wenn man einer Methode einen Primitiven-Da kann man sich sicher sein dass er nicht veränd

Übergibt man jedoch eine Klasse, können die Objektes verändert werden.

Bsp:

```
void doSome thi ng(int num) { num
int num = 5;

doSome thi ng( num);

System.ou t.p rin tln (num); // Im
r
```

Aber mit einer Klasse:

```
class MyClass { int num = 4; }

void doSome thi ng( MyClass obj) {
; }

MyClass obj = new MyClass();

doSome thi ng( obj);

System.ou t.p rin tln (ob j.num);
```




Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

## Vererbung

Vererben einer Klasse/Interfaces heißt, dass die Klasse alle Methoden/Variablen hat, von welchem sie erbt.
Dies sieht dann so aus:

**Mutter-Klasse:**

```
class A {

  public int variable = 2;

  public void test(){ System.ou t.p rin tln ("Hallo A") }

}
```

**Klasse welche von A erbt:**

```
class B extends A {

  @Ov erride // Methode von A übersc hreiben
  public void test(){ System.ou t.p rin tln ("Hallo B"); }

}
```

**Code Beispiel**

```
A obj = new A();
obj.te st(); // GIbt " Hallo A" aus
obj = new B(); // Wir können die Variable " obj " auf ein Objekt vom Typ " B" setzten, weil B von A erb
t
obj.te st(); // Gibt " Hallo B" aus
obj.va riable = 4; // Und weil " obj " immernoch vom Typ " A" ist, hat es auch immer noch seine Variabl
en
```

Zudem kann die Klasse, welche erbt, die Methoden auf ein anderes Verhalten umschreiben. (Den Code der Methode umschreiben)



Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com