

Konzept der Objektorientierung

- erlaubt uns das bündeln von zusammengehörigen Attributen und Methoden
 - wird in sogenannten Klassen gebündelt
 - Klassen dienen als Bauplan für Objekte
- => man versucht Objekte aus der realen Welt im Programm nach zu modellieren

Klassen und Objekte

- für jedes Objekt, das wir nachmodellieren möchten, müssen wir eine eigene Klasse erstellen
- Klasse stellt nur einen Bauplan da (mit dem Erstellen einer Klasse allein wird noch kein Speicherplatz belegt)
- man muss von der Klasse zunächst ein Objekt erzeugen, damit man mit diesem arbeiten kann
- new-Operator sorgt dafür, dass genügend Speicherplatz für alle Attribute innerhalb des Objekts reserviert wird
- über den Punkt Operator kann man auf die einzelnen Attribute einer Klasse zugreifen

```
public class Program {
```

```
    public static void main(String[] args) {
```

```
        //Deklaration einer Variable
        int variable;
```

```
        //Objekt von der Klasse Car instanziiieren
        Car car1 = new Car();
```



```
        car1.carBrand = "Audi";
        car1.horsePower = 250;
        car1.yearOfConstruction = 2010;
        car1.color = "blau";
```

```
        System.out.println(car1.carBrand);|
```

```
public class Car {

    String carBrand;
    int horsepower;
    int yearOfConstruction;
    String color;

}
```

 Console 

```
<terminated> Program [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Cont
Audi
```

Methoden in Klassen

- bisher besteht die Klasse Car nur aus Attributen
- dadurch sind alle Objekte, die man von der Klasse Car erstellt, sehr statisch
- damit Klassen auch Funktionalitäten besitzen, muss man Methoden implementieren

```
public class Program {

    public static void main(String[] args) {

        Car car1 = new Car();

        car1.carBrand = "Audi";
        car1.horsePower = 250;
        car1.yearOfConstruction = 2010;
        car1.color = "blau";
        car1.xPosition = 100;
        car1.yPosition = 100;

        car1.printPosition();
        car1.drive(5, 10);
        car1.printPosition();

    }

}
```

```

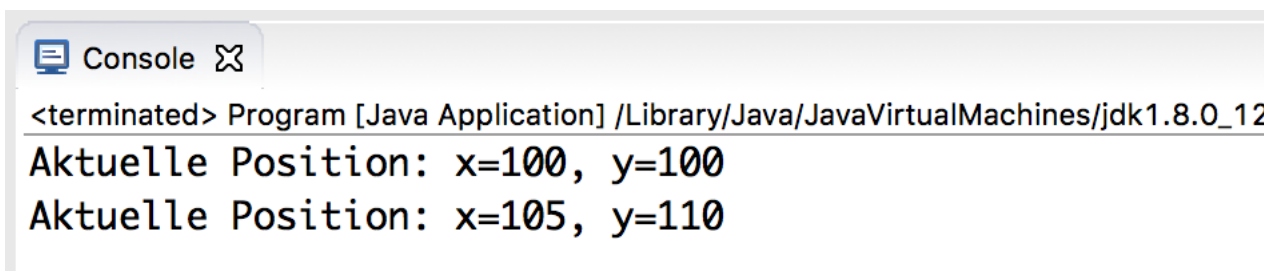
public class Car {

    String carBrand;
    int horsePower;
    int yearOfConstruction;
    String color;
    int xPosition;
    int yPosition;

    public void drive(int x, int y) {
        xPosition += x;
        yPosition += y;
    }

    public void printPosition() {
        System.out.println("Aktuelle Position: x=" + xPosition + ", y=" + yPosition);
    }
}

```



The screenshot shows a console window titled "Console" with a close button. The output text is as follows:

```

<terminated> Program [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_12
Aktuelle Position: x=100, y=100
Aktuelle Position: x=105, y=110

```

-alle Funktionen eines Autos werden also innerhalb dieser Klasse Car mithilfe von Methoden modelliert

Der Konstruktor

- Konstruktor konstruiert das Objekt
- Konstruktor deklariert alle Attribute auf dem vom new-Operator reservierten Speicherbereich
- Konstruktor ist eine besondere Methode
- falls man selbst keinen Konstruktor definiert, wird seitens Java automatisch der sogenannte Standardkonstruktor bereitgestellt
- Konstruktor hat immer den gleichen Bezeichner, wie die Klasse selbst
- Konstruktor hat keinen Rückgabewert (**Wichtig:** das Schlüsselwort void muss nicht angegeben werden)
- über Konstruktor kann man Attribute direkt mit einem gewünschten Wert vorbelegen

```
public class Car {  
  
    String carBrand;  
    int horsepower;  
    int yearOfConstruction;  
    String color;  
    int xPosition;  
    int yPosition;  
  
    Car() {  
        xPosition = 100;  
        yPosition = 100;  
    }  
}
```

Wichtig: sobald man in der Klasse einen eigenen Konstruktor definiert, wird nicht mehr automatisch der Standardkonstruktor aufgerufen

Was bedeutet „null“?

- null wird immer dann zugewiesen, wenn einer Objektvariable noch kein Objekt zugewiesen wurde
- das Wort „null“ bedeutet „nichts“ (es ist also noch kein Objekt darin enthalten)

Der Konstruktor eröffnet weitere Möglichkeiten

- bisher war der Konstruktor sehr statisch, denn es wurden immer die gleichen Werte an jedes Objekt zugewiesen
- man kann dem Konstruktor nun aber auch Übergabeparameter mitgeben, sodass man jedem Objekt individuelle Werte zuweisen kann

Beachte: Sobald man Übergabeparameter an den Konstruktor übergibt, gibt es den Standardkonstruktor nicht mehr
=> man **muss** jetzt also Parameter übergeben, ansonsten kommt es zu einem Fehler

Lösung: Den Konstruktor überladen

```
public class Program {
```

```
    public static void main(String[] args) {
```

```
        Car car1 = new Car(30, 40);  
        car1.printPosition();
```

```
        Car car2 = new Car();  
        car2.printPosition();
```

```
    }
```

```
}
```

```
public class Car {
```

```
    String carBrand;  
    int horsepower;  
    int yearOfConstruction;  
    String color;  
    int xPosition;  
    int yPosition;
```

```
    Car() {  
        xPosition = 10;  
        yPosition = 10;  
    }
```

```
    Car(int x, int y) {  
        xPosition = x;  
        yPosition = y;  
    }
```

 Console 

<terminated> Program [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.j

Aktuelle Position: x=30, y=40

Aktuelle Position: x=10, y=10

Die Sichtbarkeitsmodifizierer public und private

- man sollte nur über Methoden auf die Attribute eines Objekts zugreifen können
- durch die Verwendung des Sichtbarkeitsmodifizierers private kann man dies realisieren
- private (privat): nur innerhalb der Klasse sichtbar
- public (öffentlich): überall sichtbar
- wir werden in Zukunft alle Attribute einer Klasse mit dem Schlüsselwort private deklarieren (sauberer Programmierstil)

Getter / Setter

- aufgrund der Datenkapselung durch die Sichtbarkeitsmodifizierer sind häufig sogenannte Getter und Setter notwendig, um beispielsweise Werte einzelner Attribute auszulesen
- im zugehörigen Video wurde dies anschaulich am Beispiel mit einem Autohaus erklärt (falls du dich nicht mehr erinnern kannst, einfach noch mal ansehen)
- eine Getter-Methode gibt den Wert eines Attributs zurück
- eine Set-Methode setzt den Wert eines Attributs

Vorteil von Getter Methoden: Werte können nicht ausversehen verändert werden

Vorteil von Set Methoden: Die Gültigkeit des eingegebenen Wertes kann direkt überprüft werden

- das folgende Bild zeigt nochmal, wie man solche Getter- bzw. Setter-Methoden implementiert:

```
public class Car {  
  
    String carBrand;  
    int horsepower;  
    int yearOfConstruction;  
    String color;  
    int xPosition;  
    int yPosition;  
  
    public int getHorsePower() {  
        return horsepower;  
    }  
  
    public void setHorsePower(int hp) {  
        horsepower = hp;  
    }  
}
```