E.Tutorial
www.et.ethz.ch

Programmieren mit Java Modul 5

# Objekte

Theorieteil

## Inhaltsverzeichnis



## Begriffe

| | | |
|---|---|---|
| Klassen | Objekt-Eigenschaft | Objekt-Methode |
| Objekte | Attribut | Referenztypen |

Autoren:

Lukas Fässler, Barbara Scheuner, David Sichau

E-Mail:

et@ethz.ch

Datum:

02 July 2023

Version: 1.1

Hash: 83a8a0c

Trotz sorgfältiger Arbeit schleichen sich manchmal Fehler ein. Die Autoren sind Ihnen für Anregungen und Hinweise dankbar!

# 1 Modulübersicht

Ziel der objektorientierten Programmierung ist es, analog zur „realen Welt“, Daten mit ihrer zugehörigen Funktionalitäten in Form von **Objekten** abzubilden. Während bei der prozeduralen Programmierung Daten und die auf ihnen durchgeführten Operationen voneinander getrennt werden, sind bei der objektorientierten Programmierung Daten und Operationen in einer einzigen Struktur, der **Klasse**, vereinigt. Eine Klasse stellt eine Beschreibung („Schablone“ oder „Bauplan“) für Objekte dar, die durch gleiche oder ähnliche Eigenschaften und Verhaltensweisen charakterisiert sind. Von einer Klasse können mehrere Objekte (Exemplare oder Instanzen) erzeugt werden.

# 2 Klassen und Objekte

**Objekte** dienen dazu, **Daten (Variablen)** und ihre dazugehörigen **Funktionalität (Methoden)** zu verwalten. Aus welchen Variablen und Methoden ein Objekt aufgebaut ist, wird in der zugehörigen Klasse festgelegt. Man kann sich die Variablen einer Klasse auch als Elemente einer Liste vorstellen, wie zum Beispiel:

| | Name | Vorname | Jahrgang | Beruf |
|---|---|---|---|---|
| p1 | Mueller | Hans | 1942 | Maurer |
| p2 | Meier | Maria | 1954 | Elektrikerin |
| p3 | Koller | Andreas | 1991 | Student |

Die Klasse (analog zur Tabellenüberschrift) gibt an, dass in den Objekten (hier die einzelnen Zeilen) die Variablen `name, vorname, jahrgang` und `beruf` abgespeichert werden sollen. Die zugehörigen Objekte `p1, p2` und `p3` sehen wie folgt aus:

| p1: Person | p2: Person | p3: Person |
|---|---|---|
| name: Mueller | name: Meier | name: Koller |
| vorname: Hans | vorname: Maria | vorname: Andreas |
| jahrgang: 1942 | jahrgang: 1954 | jahrgang: 1991 |
| beruf: Maurer | beruf: Elektrikerin | beruf: Student |

Objektmethoden arbeiten mit diesen Informationen. Über eine Methode kann man zum Beispiel die Informationen über eine Person abfragen oder die Informationen updaten. Dabei gilt, dass die Methoden für jedes Objekt existieren und deshalb mit den Variablenwerten (Zustände) des zugehörigen Objekts arbeiten.

## 2.1 Klassen

Eine Klasse enthält die Vorschrift, was alles zu einem Objekt gehört. In unserem Fall sind dies Name, Vorname, Jahrgang und Beruf. Eine Klasse wird mit dem Befehl `class` definiert.

**Schreibweise:**

```
public class Person {
    // Eigenschaften des Objekts (Objektvariablen)
    // Funktionalität des Objekts (Objektmethoden)
}
```

## 2.2 Objektvariablen und Methoden

Objektvariablen und Methoden definieren alle Elemente, welche in jedem Objekt vorkommen. Diese werden alle nicht `static` deklariert und dadurch an ein Objekt und nicht an die Klasse gebunden. Die Objektmethoden, meist einfach als Methoden bezeichnet, arbeiten mit diesen Variablen. Meistens sind die Methoden dazu da, die Variablen zu verändern, ihren Wert bekannt zu geben, oder aufgrund von mehreren Variablen einen Zustand zu berechnen.

### 2.2.1 Objektvariablen (Attribute)

Die **Attribute** sind die Eigenschaften, welche in einem Objekt gespeichert werden. In diesen Variablen kann der Zustand eines Objekts gespeichert werden. Unsere Klasse `Person`, welche die Vorschrift für ein Personenobjekt enthält, sieht dann wie folgt aus:

```
public class Person {
    String name;
    String vorname;
    int jahrgang;
    String beruf;
}
```

### 2.2.2 Objektmethoden

Klassen können nicht nur Daten in Form von Variablen speichern, sondern auch Funktionalitäten anbieten. Diese Funktionalität ermöglicht die Verarbeitung der Daten.

Für die Klasse `Person` kann z.B. eine Methode angeboten werden, welche die Informationen zu einer Person ausgibt (`print`). Eine weitere Methode gibt das Alter der Person zurück, wenn das aktuelle Jahr übergeben wird (`altersAusgabe`):

```java
public class Person {
    String name;
    String vorname;
    int jahrgang;
    String beruf;

    void print(){
        System.out.println(name +"/"+ vorname);
        System.out.println(jahrgang +"/"+ beruf);
    }

    int altersAusgabe(int jahr){
        int alter = jahr-jahrgang;
        System.out.println(alter);
    }
}
```

Einer Objektmethode stehen ausser den Parametern, welche der Methode übergeben werden, auch noch die Objektvariablen (Attribute) zur Verfügung.

## 2.3 Erstellen von Objekten unter Verwendung einer Klasse

Damit man eine Person verwenden kann, muss als erstes ein Objekt dieses Typs erzeugt werden. So deklariert und erzeugt man beispielsweise das Objekt `p1` vom Typ `Person`:

**1. Deklaration:** Referenz auf die Klasse `Person` setzen und mit `p1` benennen:

```java
Person p1;
```

Im Unterschied zu primitiven Typen ist der Standardwert `null` anstelle von `0`.

**2. Erzeugung:** Mit `new` eine Instanz der Klasse Person erzeugen und in der Referenz `p1` speichern:

```java
p1 = new Person();
```

Meistens werden die beiden Schritte in einer Anweisung geschrieben:

```java
Person p1 = new Person();
```

Das Erzeugen von Objekten geschieht meist in einer weiteren Klasse, welche ihre Daten in der angebotenen Klasse `Person` speichern möchte. Die Klasse `Personalverwaltung` möchte zum Beispiel neue Personen-Objekte erzeugen können. Angenommen, diese Personalverwaltung sei der Startpunkt unseres Programms, dann könnte diese z.B. so aussehen:

```java
public class Personalverwaltung {
    public static void main(String[] args){
        // Deklaration
        Person p1;
        // Erzeugung eines neuen Objektstion
        p1 = new Person();
        // Zuweisung von Werten für die Attribute
        p1.name = "Müller";
        p1.vorname = "Andreas";
        p1.jahrgang = 1960;
        p1.beruf = "Maurer";
    }
}
```

### 2.3.1 Unterschiede zwischen primitiven Datentypen und Referenztypen

Wenn eine Variable ein Objekt verwalten soll, dann nennt man diese Variable eine **Referenzvariable**. Bei der Deklaration gibt es zwischen Referenztypen und primitiven Datentypen (int, char, double, etc.) keine Unterschiede. Bei der Initialisierung hingegen gibt es Unterschiede. Referenztypen halten Referenzen auf Objekte und bieten ein Mittel, um auf die Objekte zuzugreifen, die irgendwo im Speicher abgelegt sind, ähnlich einem Link oder einer Verknüpfung. Die Speicherorte selbst sind für das Programmieren irrelevant.

**Beispiel:** Angenommen, wir möchten bei unserem Beispiel eine neue Person `p2` erzeugen (dies würde dem Einfügen einer neuen Zeile in der Tabelle entsprechen), reicht es nicht, die Referenz zu kopieren. Folgendes Beispiel demonstriert, weshalb wir zunächst per `new`-Operator ein neues Objekt erzeugen müssen, bevor wir die Werte der Variablen neu setzen können.

Gegeben seien:

```java
int wert = 8;
Person p1 = new Person();
```

Der aktuelle Speicher präsentiert sich wie in Abbildung 1:

Im Speicher des primitiven Datentyps befindet sich eine 8, die zum Variablennamen `wert` gehört. Beim Referenztyp befindet sich eine Referenz auf ein Personenobjekt, welche unter dem Variablennamen `p1` erreichbar ist. Im Speicherbereich der Person ist zu sehen, dass nur beim Jahrgang ein Wert gespeichert ist. Dies ist so, weil String ebenfalls eine Klasse bezeichnet (beachten Sie die Grossschreibung). Ist noch nicht bestimmt, auf welchen Speicherbereich sich eine Referenzvariable bezieht, wird `null` initialisiert.

Mit folgenden Zeilen werden die Attribut-Werte zugewiesen:

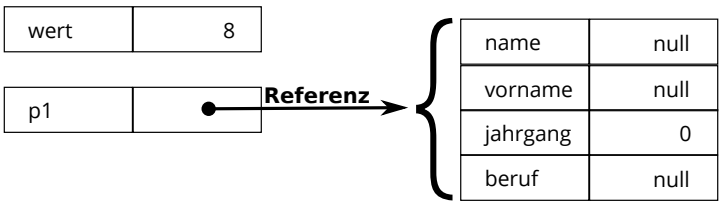


Abbildung 1: Speicher nach Erzeugung eines neuen Objektes.

```
p1.name = "Meier";
p1.vorname = "Hans";
p1.jahrgang = 1960;
p1.beruf = "Maler";
```

Der Speicher ändert sich wie in Abbildung 2 dargestellt.

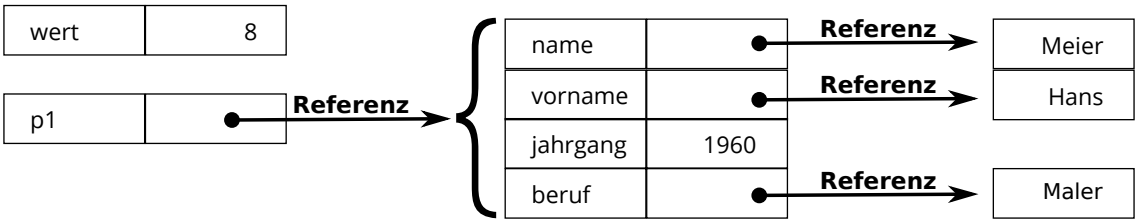


Abbildung 2: Speicher nach Zuweisung der Attributswerte.

Nun führen wir folgende Zeilen aus:

```
int wert2 = wert;
Person p2 = p1;
```

Der Speicher präsentiert sich nun wie in Abbildung 3 dargestellt.

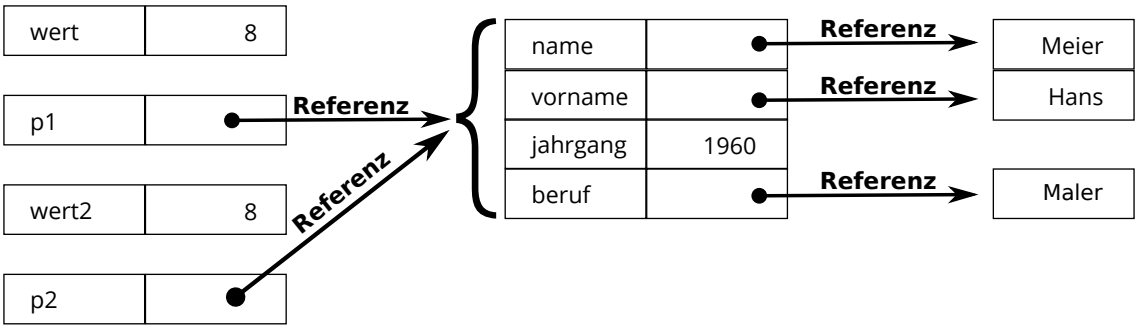


Abbildung 3: Speicher nachdem `p2` mit `p1` initalisiert wurde.

Beim primitiven Datentyp wird neuer Speicher bereitgestellt, der Wert wird kopiert. Bei der Referenzvariablen wird die **Referenz** kopiert, nicht aber das ganze Objekt. Werden

nun die folgenden Zeilen ausgeführt, sieht der Speicher anschliessend wie in Abbildung 4 aus.

```
wert2 = 10;
p2.jahrgang = 1967;
```

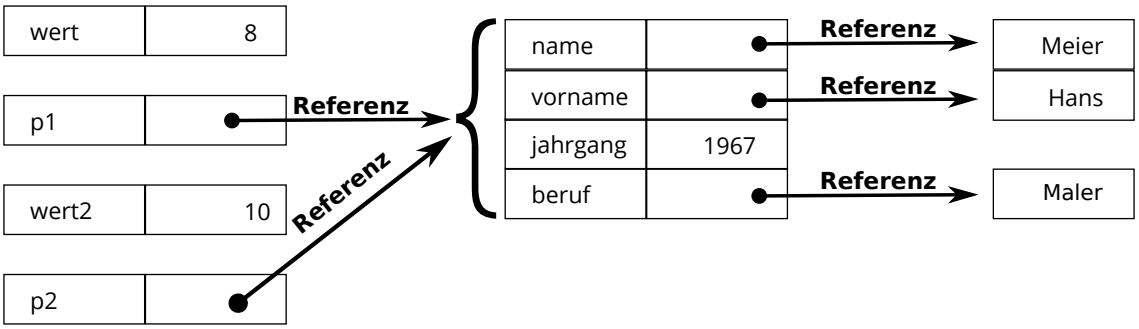


Abbildung 4: Speicher nachdem das Attribut p2 verändert wurde.

Folgende Tabelle fasst die Unterschiede zwischen Referenztypen und primitive Typen zusammen.

| | Primitive Datentypen | Referenztypen |
|---|---|---|
| Anzahl | Umfasst die Typen boolean, char, byte, short, long, float und double | Die Anzahl von Referenztypen ist unbegrenzt, da diese vom User definiert werden. |
| Default-Wert | 0 | null |
| Speicherort | Der Speicherort speichert die tatsächlichen Daten, die der primitive Typ enthält. | Der Speicherort speichert eine Referenz auf die Daten. |
| Zuweisung an eine andere Variable | Wird der Wert eines primitiven Typs einer anderen Variablen zugewiesen, wird eine Kopie erstellt. | Wird einem Referenztyp ein anderer Referenztyp zugewiesen, zeigen beide auf das gleiche Objekt. |
| Übergabe an Methode | Wird einer Methode ein primitiver Typ übergeben, wird nur eine Kopie des primitiven Werts übergeben. Die aufgerufene Methode hat keinerlei Zugriff auf den ursprünglichen primitiven Wert und kann ihn deswegen nicht ändern. Die aufgerufene Methode kann den kopierten Wert ändern. | Wird eine Methode ein Objekt übergeben, kann die aufgerufene Methode den Inhalt des Objekts ändern, nicht aber seine Adresse. |