



<a href="http://ovm-kassel.de">http://ovm-kassel.de</a>   Information	
Information IT-AE-UML-INFO-3.1 Objektorientierte Analyse (OOA)	
Code	IT-AE-UML-INFO-3.1
Autor	André Bauer <a(dot)bauer(at)ovm-kassel(dot)de>
Datum	10. März 2018
Links	<a href="#">Wikipedia: Objektorientierte Analyse und Design</a>
Verwandte Literatur	<ul style="list-style-type: none"> <li>• IT-AE-UML-INFO-3.2</li> <li>• IT-AE-UML-INFO-3.3</li> </ul>
Lernjobs	<ul style="list-style-type: none"> <li>• IT-AE-UML-LJ-3.1 bis 3.4</li> </ul>
Lizenz	 <p>Dieses Werk ist lizenziert unter einer <a href="#">Creative Commons Namensnennung - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz</a>.</p>

## Objektorientierte Analyse (OOA)

Wenn eine Software mehr als etwa hundert Zeilen Quellcode umfasst, spätestens ab mehreren hundert, benötigt man Techniken, um der Software eine klare Struktur zu geben. Eine klare Struktur ist wichtig, damit die Software leicht angepasst und erweitert werden kann, sowohl durch bisherige als auch durch andere Entwickler.

### 1. User-Stories

Eine sehr häufig angewandte Technik zum Entwurf und zur Strukturierung ist die objektorientierte Software-Entwicklung. Dazu wird in der objektorientierten Analyse anhand von Szenarien ermittelt, wie die zukünftige Nutzung des (Software-)Systems ablaufen soll. Dabei werden sogenannte [User-Stories](#) verfasst:

#### User-Story 1

Die Lampen lampe1, lampe2 und lampe3 sind anfangs ausgeschaltet. Der Schalter schalter1 ist ebenfalls in der Position „aus“. Der Benutzer betätigt den Schalter schalter1. Die Lampen lampe1, lampe2 und lampe3 werden eingeschaltet.

Aus den User-Stories können nun die relevanten Objekte abgeleitet werden; im Beispiel sind dies

der Benutzer, der Schalter schalter1 sowie die Lampen lampe1, lampe2 und lampe3.

## 2. Das Verfahren nach Abbott

Mit dem Verfahren von Abbott werden aus einer Problem- oder Situationsbeschreibung oder entsprechenden User-Stories mögliche Objekte, Attribute und Methoden ermittelt. Dieses Verfahren wird auf dem Informationsblatt IT-AE-JA-INFO-5.2 erläutert.

## 3. Sequenzdiagramme

Stellt man die User-Story in einem **UML-Sequenzdiagramm** dar, so kann man den Programmablauf entlang der Pfeile ablesen. So sendet der Schalter schalter1 die Nachricht `einschalten()` an die Lampe lampe1. Erst wenn diese den Einschaltvorgang abgeschlossen hat, setzt schalter1 seine Arbeit fort und sendet lampe2 die Nachricht `einschalten()` usw.

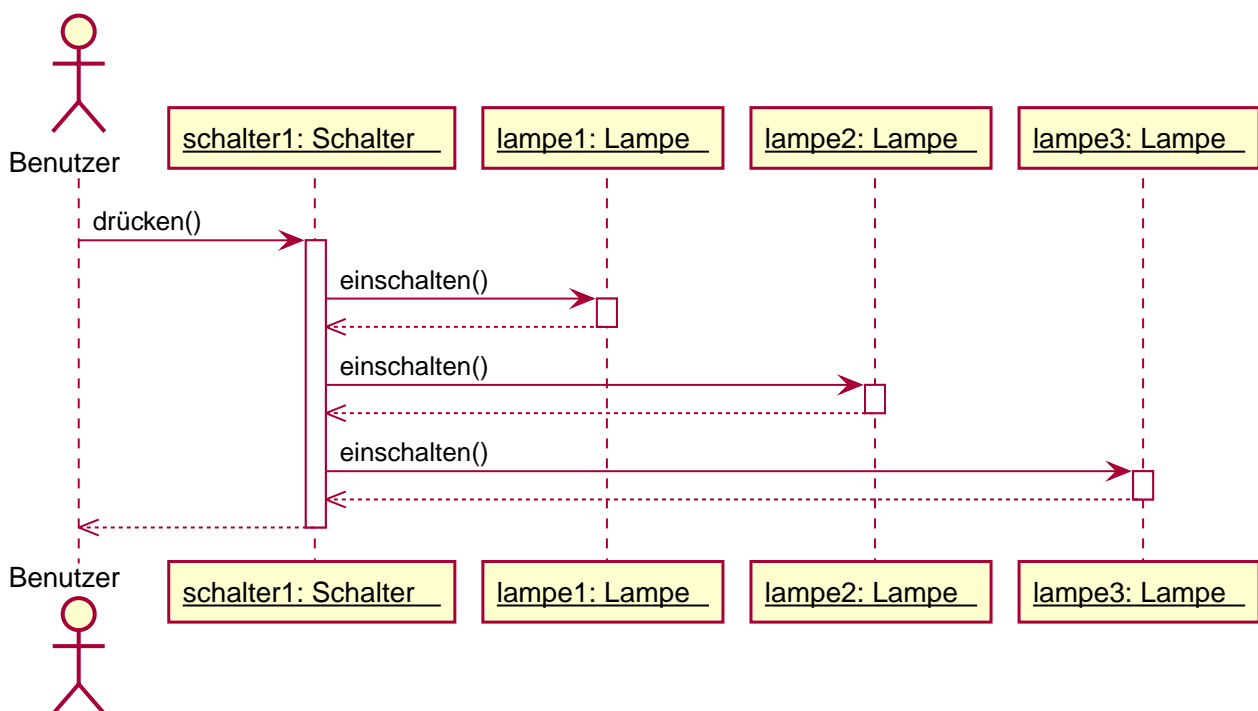


Abbildung 1. Sequenzdiagramm zur User-Story 1

Eine zweite User-Story vervollständigt die möglichen Anwendungsfälle der Schaltung:

### User-Story 2

Die Lampen lampe1, lampe2 und lampe3 sind eingeschaltet. Der Schalter schalter1 ist ebenfalls in der Position „ein“. Der Benutzer betätigt den Schalter schalter1. Die Lampen lampe1, lampe2 und lampe3 werden ausgeschaltet.

Daraus ergibt sich das folgende Sequenzdiagramm:

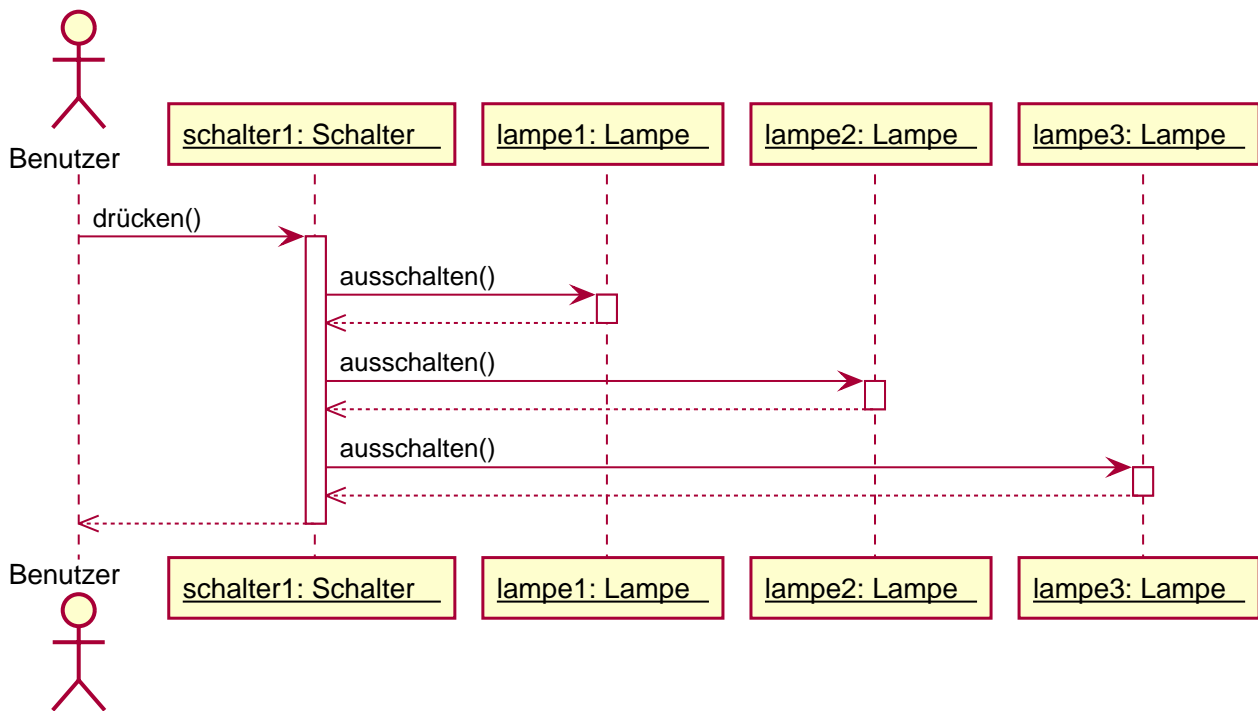


Abbildung 2. Sequenzdiagramm zur User-Story 2

## 4. Objektdiagramme

Aus den User-Stories bzw. den zugehörigen Sequenzdiagrammen kann man ablesen, wie die beteiligten Objekte miteinander verknüpft sein müssen, um Nachrichten austauschen zu können.

Objekte, die Nachrichten austauschen, werden im Diagramm mit Linien verbunden. Dieser Beziehungstyp wird *Assoziation* genannt (auch Kennt-Beziehung). Die Beziehungen werden mit einem Namen versehen, im Beispiel lautet dieser „schaltet“. Wenn die Nachrichten nur in eine Richtung ausgetauscht werden, wird dies durch eine Pfeilspitze kenntlich gemacht.

Der Schalter schalter1 verwendet die Lampen lampe1, lampe2 und lampe3, um ihnen jeweils die Nachricht  `einschalten()` bzw.  `ausschalten()` senden zu können. Diese Beziehung wird daher in dem Objektdiagramm in Abbildung 3 durch Pfeile dargestellt.

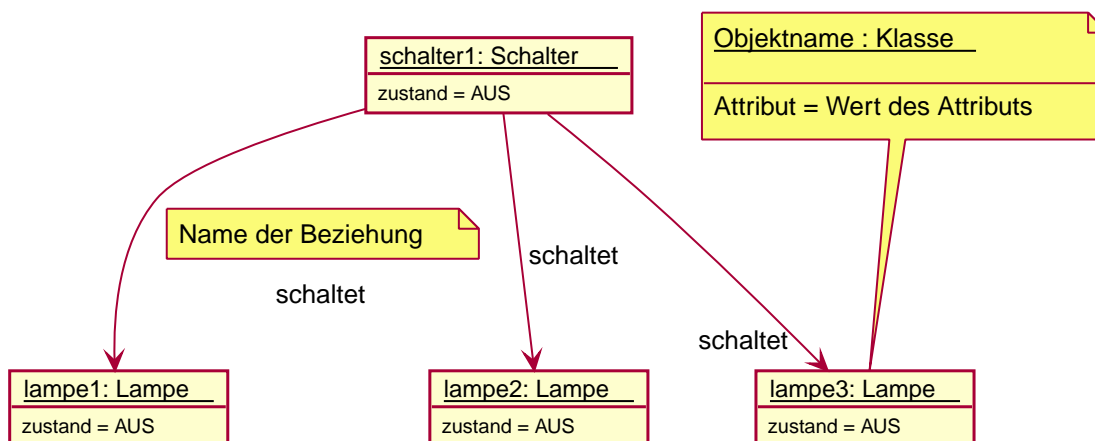
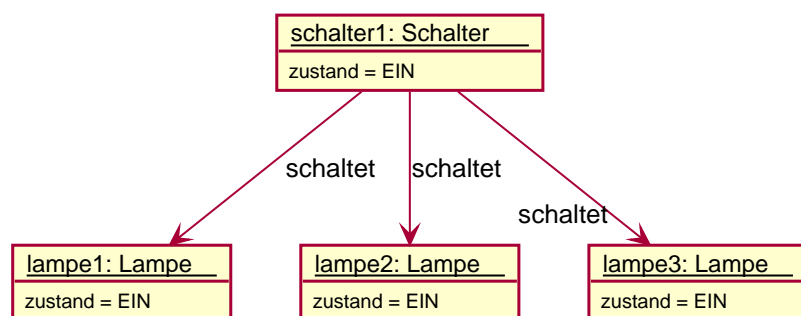


Abbildung 3. Objektdiagramm für einen Aufbau mit einem Schalter und drei Lampen zu Beginn der User-Story 1

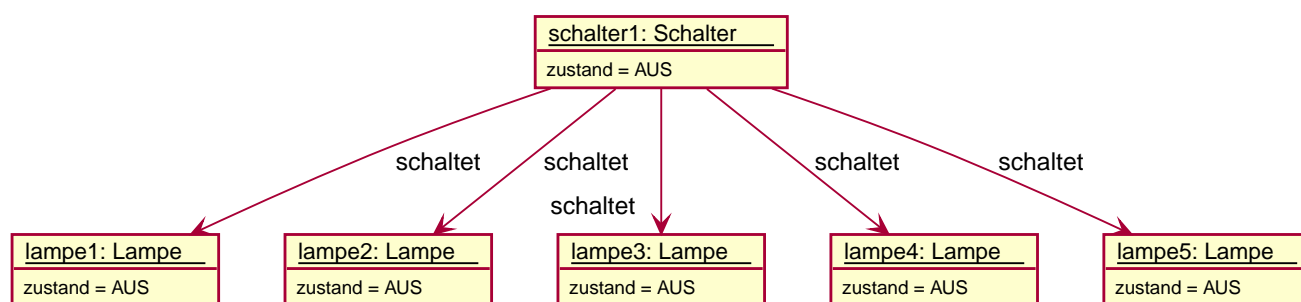
Ein Objektdiagramm stellt den Zustand eines Software-Systems zu einem bestimmten Zeitpunkt dar. Das Diagramm in [Abbildung 3](#) stellt den Zustand zu Beginn der [User-Story 1](#) bzw. zum Ende der [User-Story 2](#) dar, da der Schalter und die Lampen den Zustand **AUS** haben.

Der Zustand zum Ende der [User-Story 1](#) bzw. zu Beginn der [User-Story 2](#) sieht so aus:



*Abbildung 4. Objektdiagramm für einen Aufbau mit einem Schalter und drei Lampen zum Ende der [User-Story 1](#)*

Man könnte jetzt den Schalter und jede der drei Lampen direkt programmieren, aber dies hätte Nachteile: für die Lampen würde man dreimal dasselbe Programm schreiben. Es wäre daher eine unflexible Lösung, man möchte mit derselben Software auch Schaltungen mit vier oder mehr Lampen mit einem Schalter verbinden können.



*Abbildung 5. Objektdiagramm für einen Aufbau mit einem Schalter und fünf Lampen*

## 5. Klassendiagramme

Daher verallgemeinert man die möglichen Objektbeziehungen in Form eines [UML-Klassendiagramms](#):

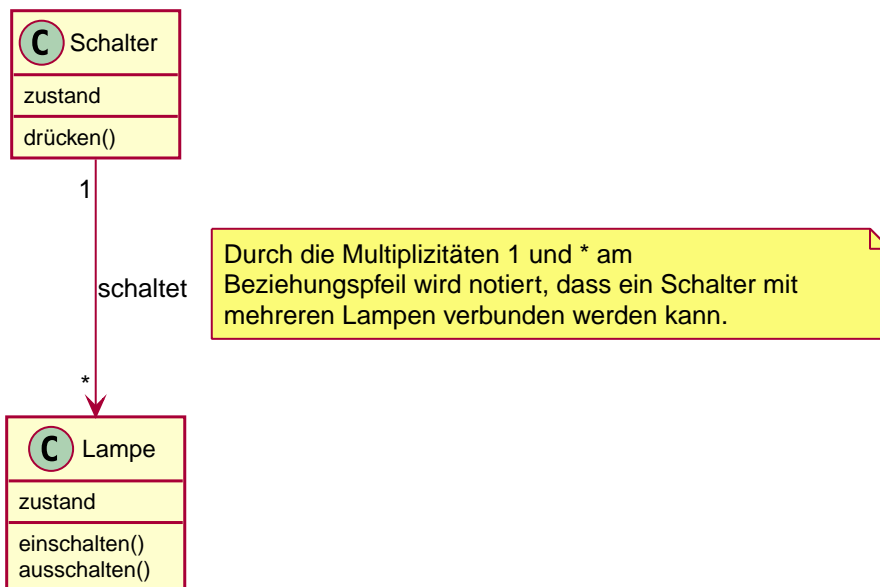


Abbildung 6. Klassendiagramm als Ergebnis der OOA

In einem Klassendiagramm werden Objekte mit derselben Struktur zu einer Klasse zusammengefasst. Die Lampen `lampe1`, `lampe2` und `lampe3` werden also zu der Klasse `Lampe` zusammengefasst. Die Nachricht `drücken()` wird dem Schalter und `ausschalten()` bei der `Lampe` notiert, also jeweils bei der Klasse, der die entsprechende Nachricht gesendet wird. Statt Nachricht wird im Zusammenhang mit der Programmiersprache Java der Begriff *Methode* verwendet. Die Daten, die auch Attribute der Objekte bzw. Klassen genannt werden, können im Klassendiagramm mit ihren initialen Werten angegeben werden, zudem kann auch der entsprechende Datentyp benannt werden wie in [Abbildung 7](#).

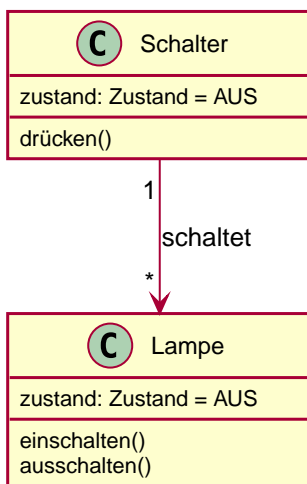


Abbildung 7. Klassendiagramm mit Datentypen und initialen Werten.

Das Klassendiagramm stellt im Gegensatz zu einem Objektdiagramm keinen Systemzustand dar, sondern die Struktur der zugrundeliegenden Software bzw. des Quellcodes. Die Darstellung einer Klasse im Klassendiagramm folgt dem folgenden Schema:

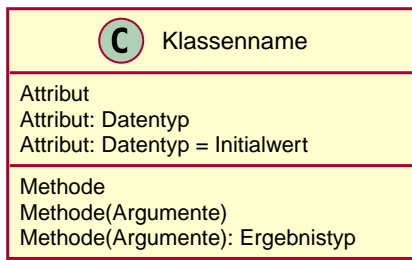


Abbildung 8. Schema der graphischen Darstellung einer Klasse