

Einführung in die Objektorientierung

Software Entwicklung 1

Annette Bieniusa, Mathias Weber, Peter Zeller

1 Motivation

“The basic philosophy underlying object-oriented programming is to make the programs as far as possible reflect that part of the reality they are going to treat.

It is then often easier to understand and to get an overview of what is described in programs.

The reason is that human beings from the outset are used to and trained in the perception of what is going on in the real world.

The closer it is possible to use this way of thinking in programming, the easier it is to write and understand programs.” [O. Lehrmann Madsen, B. Møller-Petersen, K. Nygaard: Object-oriented Programming in the BETA Programming Language, Addison-Wesley, 1993]

Das Paradigma der Objektorientierung bezieht seine konzeptionellen Grundlagen aus der realen Welt. Für den Menschen besteht die physische/materielle Umgebung und die gedankliche/geistige Welt aus logisch zusammengehörigen Objekten mit

- eigenständiger Identität
- einem Zustand, der sich mit der Zeit ändern kann,
- der Möglichkeit auf sie einwirken zu können bzw. der Fähigkeit zu kommunizieren und zu kooperieren.

1.1 Charakterisierung von Objekten

Anhand der charakteristischen Eigenschaften von Objekten (Zustand, Lebensdauer, Ort, Verhalten und Identität) lässt sich untersuchen, wie ausgeprägt der Objektcharakter eines Gegenstands ist. Die folgenden Erläuterungen sind als Hilfestellung zu verstehen, nicht als scharfe Begriffsklärung.

Zustand Objekte haben *Attribute* (Alter, Größe,...). Die *Werte der Attribute* können sich verändern; in der Programmierung heißt das, dass jedes Objekt für jedes Attribut eine Variable besitzt. Der *Zustand eines Objekts* zu einem Zeitpunkt ist charakterisiert durch die aktuellen Attributwerte.

Lebensdauer Jedes Objekt hat eine Lebensdauer. Dies ist die Zeit von seiner Entstehung bis zum Verschwinden bzw. von seiner Erzeugung bis zur Löschung. Die Lebensdauer kann in Zeit oder in Ablaufschritten gemessen werden.

Aufenthaltort Objekte besitzen normalerweise einen Ort, der durch eine Adresse charakterisiert wird. Beispiele für Adressen: Hausadresse, E-Mail-Adresse, Web-Adresse, Telefonnummer, Geo-Koordinaten, Rechneradresse, Adresse im Hauptspeicher.

Verhalten Objekte können

- ihren Zustand verändern,
- ihren Aufenthaltsort verändern,
- anderen Objekten eine Nachricht schicken,
- Nachrichten von anderen Objekten empfangen,
- neue Objekte erzeugen.

Zu jeder Nachricht gibt es eine sogenannte Methode, die beschreibt, wie eine Nachricht bearbeitet wird.

Identität Zu einem Objekt gibt es im Allgemeinen gleichartige Objekte (z.B. zwei Bücher, zwei Häuser,...) Objekte kann man miteinander vergleichen. Sie besitzen eine Identität, die vom Zustand unabhängig ist; d.h. sie können sich in allen Eigenschaften gleichen, ohne identisch zu sein (z.B. zwei baugleiche Autos). Insbesondere kann man durch Klonen Objekte erzeugen, die sich in allen Eigenschaften gleichen, aber nicht identisch sind.

Beispiel Wir erläutern die Objektcharakteristiken am Beispiel eines Autos:

- Attribute und ihr Zustand:

Farbe:	rot
Marke:	VW
Schiebedach:	ja
Alter:	12 Jahre
Tankfüllung:	halbvoll
Kilometerstand:	234568
Geschwindigkeit:	121 km/h

- Lebensdauer: seit 1996 bis ??
- Ort: Autobahn A1 zwischen Ennepetal und Volmarstein
- Verhalten, d.h. Nachrichten, die verstanden werden:
 - bremsen
 - Gas geben
 - Scheibenwischer (an/aus)
 - ...
- Identität: gegeben.

Frage 1: Haben die folgenden Dingen Objektcharakter, d.h. haben sie eine Identität, Zustand, Interaktionsmöglichkeiten?

- Häuser
- Webserver
- Länder
- Liebe
- Vorlesungen
- Demokratie
- Zahlen und Funktionen

Abgrenzung: Werte vs. Objekte

Die mathematischen “Objekte” (Funktionen, Mengen, Zahlen, etc.) besitzen

- keinen (veränderlichen) Zustand,
- keine Lebensdauer,
- keinen Aufenthaltsort,
- kein Verhalten,
- keine Identität jenseits der Gleichheit.

Deshalb nennen wir sie *Werte*. (Allerdings können Werte durch Objekte repräsentiert werden.)

1.2 Beziehungen zwischen Objekten

Objekte können zueinander in unterschiedlichen Beziehungen stehen:

- Objekt X kann ein Teil von Objekt Y sein.
- Objekt X kann mit einem anderen Objekt Y assoziiert oder verknüpft sein.

Besteht keine Beziehung zwischen zwei Objekten, können sie auch nicht direkt miteinander kommunizieren.

1.3 Nachrichten und Methoden

Objekte *bieten Dienste an* und *nehmen Dienste* anderer Objekte *in Anspruch*.

Um einen Dienst in Anspruch zu nehmen, schickt ein Objekt einem anderen eine **Nachricht**, die den Dienst bezeichnet und Parameter übergibt (Auftragserteilung). Erhält ein Objekt eine Nachricht, führt es eine **Methode** aus, die der Handlungsvorschrift zur Ausführung des Dienstes entspricht. In der Objektorientierung werden also Auftragserteilung und -ausführung getrennt.

Beispiel: Dienste und Aufträge Wir modellieren hier einen Buchkauf und die involvierten Dienste bzw. Aufträge:

- Buchhändlerin B bietet den Dienst an, per E-Mail Bücher zu bestellen.
- Herr K. (Senderobjekt) gibt Frau B. (Empfängerobjekt) den Auftrag, das Werk "Per Anhalter durch die Galaxis" zu besorgen.
- Herr K. weiß nicht, wie Frau B. den Auftrag ausführt.
- Frau B. besitzt eine Methode, wie mit dem Auftrag zu verfahren ist. Nach Ausführung schickt sie Herrn K. das Buch zu.

1.4 Objektbasiertes System

In der Objektorientierung werden Systeme als Menge kooperierender Objekte modelliert, die untereinander in Beziehung stehen und über Nachrichten miteinander kooperieren. Derartige Systeme heißen **objektbasiert**.

Grundsätzlich gibt es zwei Konzepte zur programmiersprachlichen Beschreibung von Objekten:

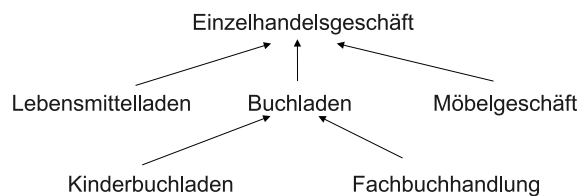
- **Prototyp-Konzept:**
Der Programmierer beschreibt direkt einzelne Objekte. Neue Objekte werden durch Klonen existierender Objekte und Verändern ihrer Eigenschaften zur Laufzeit erzeugt.

- **Klassenkonzept:**

Der Programmierer deklariert Klassen als Beschreibung der Eigenschaften, die Objekte dieser Klasse haben sollen. Die Programmiersprache ermöglicht es, zur Laufzeit Objekte der Klassen zu erzeugen, aber nicht, die Klassen zu verändern.

Wir betrachten in SE1 nur das Klassenkonzept, da dieses von Java implementiert wird. Objekte lassen sich nach ihren Eigenschaften **klassifizieren**. D.h. alle Objekte mit ähnlichen Eigenschaften werden zu einer Klasse zusammengefasst. Die Klassen werden außerdem hierarchisch geordnet. Die übergeordneten Klassen besitzen Eigenschaften, die allen untergeordneten Klassen gemeinsam sind.

Beispiel Als Objekte betrachten wir Einzelhandelsgeschäfte. Sie lassen sich nach ihren Produkten klassifizieren:



Es gibt Klassen, zu denen nur die Objekte der Unterklassen gehören (z.B. gibt es kein Geschäft, das nur ein Einzelhandelsgeschäft ist). Diese Klassen nennt man **abstrakt**. Es gibt Klassen, zu denen eigene Objekte und die Objekte der untergeordneten Klassen gehören (z.B. Buchladen).

Die Vorteile einer Klassifikation liegen auf der Hand:

- Übergeordnete Klassen besitzen Eigenschaften, die allen untergeordneten Klassen gemeinsam sind. Auf Basis dieser Eigenschaften lassen sich Methoden formulieren, die für alle Objekte der untergeordneten Klassen funktionieren.
- Eigenschaften können von übergeordneten zu untergeordneten Klassen **vererbt** werden.

Objektbasierte Systeme, bei denen die Objekte Klassen zugeordnet und die Klassen gemäß einer Klassifikation hierarchisch geordnet sind und Vererbung erlauben, heißen **objektorientiert**.

1.5 Zusammenfassung

Die Eigenschaften und das Verhalten eines (programmiersprachlichen) Objekts ergeben sich aus seinen möglichen Zuständen und daraus, wie es auf Nachrichten reagiert. Eine Objektbeschreibung - insbesondere eine Klassendeklaration - muss daher festlegen:

- welche Zustände ein Objekt annehmen kann,

- auf welche Nachrichten es reagieren kann und
- wie die Methoden aussehen, mit denen ein Objekt auf den Empfang von Nachrichten reagieren kann.

Die Menge der möglichen Zustände eines Objekts entspricht den Wertebereichen seiner Attribute. Die Reaktionen eines Objekts auf eintreffende Nachrichten legen sein dynamisches Verhalten fest.

2 Objektorientierte Programmierung in Java

Java ist eine objektorientierte Programmiersprache; d.h. Objekte und Klassen sind native Abstraktionsmechanismen bei der Datenmodellierung und -bearbeitung.

Wir erinnern uns: Datentypen bestehen aus einer Wertemenge und Operationen, die für diese Werte definiert sind. Neben den primitiven Datentypen (`boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, `short`) gibt es in Java **Referenztypen**. Sie bilden die Grundlage der objektorientierten Programmierung in Java. Ein Objekt nimmt den Wert eines Referenzdatentyps an. Java selbst bietet eine Vielzahl an Bibliotheken mit den unterschiedlichsten Referenztypen an. Wir werden in diesem Abschnitt eine davon näher betrachten: `Color`.

In diesem Abschnitt liegt der Fokus auf der Verwendung von Referenztypen, insbesondere Objekterzeugung und Methodenaufruf. Der Entwurf und die Implementierung eigener Datentypen behandeln in der nächsten Vorlesung.

2.1 Referenztypen

Um einen Datentypen zu verwenden, muss man nicht wissen, wie er implementiert ist. Wir haben beispielsweise die primitiven Datentypen oder auch Strings verwendet, ohne deren tatsächliche Implementierung zu kennen.

Auch für Referenztypen können wir die Informationen, die nötig sind, um den Datentypen zu verwenden, in Form einer Programmierschnittstelle (Application Programming Interface, kurz API) bereitstellen und so von der eigentlichen Implementierung abstrahieren.

Referenztypen werden unter anderem in Form von Klassen implementiert. Der Klassenname wird gleichzeitig als *Typname* für die Objekte dieser Klasse verwendet (*Klassentyp*). Er kann im Programm dann wie elementare Typen (`int`, `double`, usw.) für die Deklaration von lokalen Variablen, Parametern und Rückgabewerten verwendet werden.

Beispiel: API von Color Die Klasse `Color` ist bereits in der Java Standardbibliothek enthalten. Um sie in einem Programm zu verwenden, kann der Datentyp mittels `import java.awt.Color`; am Anfang der Datei importiert werden. Eine Farbe ist durch drei `int`-Werte zwischen 0 und 255 beschrieben, welche die Intensität der Farben Rot, Grün und Blau angeben (255 ist am hellsten).

Rot	Grün	Blau	Farbeton
255	0	0	Rot
0	255	0	Grün
0	0	255	Blau
255	255	0	Gelb
0	0	0	Schwarz
255	255	255	Weiß
100	100	100	Grau

Hier ein Auszug aus der API für den Datentyp `Color`:

API von <code>java.awt.Color</code>		
	<code>Color(int r, int g, int b)</code>	Konstruktor
<code>int</code>	<code>getRed()</code>	Intensität für Rot
<code>int</code>	<code>getGreen()</code>	Intensität für Grün
<code>int</code>	<code>getBlue()</code>	Intensität für Blau
<code>Color</code>	<code>brighter()</code>	Hellere Version dieser Farbe
<code>Color</code>	<code>darker()</code>	Dunklere Version dieser Farbe
<code>String</code>	<code>toString()</code>	String-Darstellung dieser Farbe
<code>boolean</code>	<code>equals(Color c)</code>	Vergleich ¹

Ein Farbe-Objekt kann über den Konstruktor `Color(int red, int green, int blue)` erstellt werden. Zum Lesen der Farbwerte gibt es Methoden `getRed`, `getGreen` und `getBlue`, welche den jeweiligen Farbwert als `int` zurückgeben. `darker` und `brighter` liefern jeweils `Color`-Objekte für dunklere bzw. hellere Versionen der Farbe. Die Methode `toString` stellt eine String-Repräsentierung des Objekts bereit und `equals` erlaubt es Farben auf Gleichheit zu prüfen.



Um den `Color`-Datentyp in unseren Programmen zu verwenden, müssen wir explizit angeben, in welcher Java-Bibliothek dieser definiert wird. Daher muss am Anfang eines Programms, das die Klasse verwendet, die Anweisung `import java.awt.Color`; eingefügt werden.

Das folgende Programm zeigt die Verwendung von `Color`-Objekten. Wir erläutern die einzelnen Konstrukte im Folgenden.

```
import java.awt.Color;

public class ColorSquares {
    public static void main(String[] args) {
        int r = Integer.parseInt(args[0]);
        int g = Integer.parseInt(args[1]);
        int b = Integer.parseInt(args[2]);

        Color c1 = new Color(r,g,b);
        Color c2 = c1.brighter();
    }
}
```

¹Genau genommen nimmt die `equals` Methode ein beliebiges Objekt, in Java repräsentiert durch den Typ `Object`. Die Klasse `Object` betrachten wir in einem späteren Kapitel. Für Objekte, die nicht vom Typ `Color` sind, gibt sie immer `false` zurück.

```

    Color c3 = c1.darker();

    StdDraw.setPenColor(c3);
    StdDraw.filledSquare(.25,.5,.3);

    StdDraw.setPenColor(c1);
    StdDraw.filledSquare(.25,.5,.2);

    StdDraw.setPenColor(c2);
    StdDraw.filledSquare(.25,.5,.1);
}
}

```

Beispiel: API von String Auch Strings sind in Java Objekte. Hier ein Auszug aus der API von `String`:

API von String	
	<code>String(char[] value)</code>
	<code>String(char[] value, int offset, int count)</code>
<code>char[]</code>	<code>toCharArray()</code>
<code>char</code>	<code>charAt(int index)</code>
<code>int</code>	<code>indexOf(int ch)</code>
<code>boolean</code>	<code>equals(String s)</code>

- Der Konstruktor `String(char[] value)` liefert ein String-Objekt mit den Zeichen des Arrays `value`.
- Der Konstruktor `String(char[] value, int offset, int count)` liefert ein String-Objekt der Länge `count` mit den Zeichen aus dem Array `value`, ab der Position `offset`.
- Die Methode `toCharArray()` liefert für ein String-Objekte ein `char`-Array, dessen Einträge den Zeichen des Strings entsprechen.
- Die Methode `charAt(int index)` liefert den Character an der Position `index` in einem String.
- Die Methode `indexOf(int ch)` liefert die Position des ersten Vorkommens des Characters `ch` in einem String.
- Die Methode `equals(String s)` wird wie bei `Color` für den Vergleich benutzt.

Beispiele für die Verwendung von Strings:

```

char[] a = {'H', 'e', 'l', 'l', 'o'};
String b = new String(a); // b ist "Hello"
String c = new String(a, 1, 2); // c ist "el"
char[] d = "World".toCharArray(); // d ist [W, o, r, l, d]
char e = "Hello".charAt(1); // e ist 'e'
int f = "Hello".indexOf('l'); // f ist 2

```



```
int g = "Hello".indexOf('x'); // g ist -1
boolean h = b == "Hello"; // h ist false
boolean i = b.equals("Hello"); // i ist true
```

2.2 Objekte erzeugen und referenzieren



Kapitel 3.1 aus R. Sedgewick, K. Wayne: Einführung in die Programmierung mit Java. 2011, Pearson Studium.

Schauen wir uns zunächst an, wie Objekte erzeugt werden können.

Syntax in Java:

Objekte werden mit Ausdrücken folgender Form erzeugt (engl. *object creation expression*):

Ausdruck \rightarrow
`new` \ll *Bezeichner* \gg (AktuelleParameterListe)

wobei

AktuelleParameterListe \rightarrow ε
 | AusdruckListe

AusdruckListe \rightarrow Ausdruck
 | Ausdruck , AusdruckListe

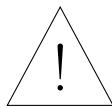
Semantik:

1. Erzeuge ein Objekt / eine Instanz der Klasse, der der Konstruktor gehört.
2. Werte die aktuellen Parameter aus.
3. Rufe den Konstruktor mit den Parametern auf. Dieser sollte die Instanzvariablen initialisieren.

Ergebnis ist die Referenz auf das neu erzeugte Objekt.

Eine **Objektreferenz** (engl. *object reference*) ist eine eindeutige abstrakte Adresse oder Bezeichnung für ein Objekt. Manchmal spricht man auch von **Verweis** (engl. *link*) oder **Zeiger** (engl. *pointer*). Über Referenzen kann man mit den referenzierten Objekten interagieren (z.B. Methoden aufrufen und auf seine Attribute zugreifen).

Die Deklaration von Referenzvariablen erfolgt genauso wie für Variablen von primitiven Typen bzw. Arrays.



- Die Deklaration einer Variable allein erzeugt kein Objekt! Dies geschieht erst durch den Konstruktoraufruf.
- Variablen von Referenztypen speichern nicht die Objekte als Ganzes, sondern Objektreferenzen.

Beispiel Von einer Klasse lassen sich beliebig viele Objekte erzeugen:

```

Color c = new Color(255,0,0);
Color d = new Color(150,150,150);
Color e = new Color(255,0,0);
Color f = d;

```

In diesem Beispiel referenzieren `c` und `e` verschiedene Objekte, die aber den gleichen Wert speichern. `d` und `f` referenzieren das gleiche Objekt.

2.2.1 Operationen auf Referenzen

Referenzen lassen sich mit `==` auf Gleichheit testen bzw. mit `!=` auf Ungleichheit. Sie sind genau dann gleich, wenn sie dasselbe Objekt referenzieren.

Beispiel:

Nach der obigen Zuweisung an `f` gilt, dass `d == f` und `c != d` und `c != e`. Objekte können der gleichen Klasse angehören und den gleichen Zustand haben (gleich sein), aber trotzdem nicht dieselbe Identität haben und damit auch unterschiedliche Referenzen besitzen.

Um Objekte auf Gleichheit zu testen, verwenden wir die `equals()`-Methode. Der Ausdruck `c.equals(e)` wertet im Beispiel zu `true` aus.

2.2.2 Die null-Referenz

Wie wir im Kapitel zu Arrays gesehen haben, ist `null` ein spezieller Referenzwert, der auf nichts verweist.

Folgendes Programmfragment illustriert die Operationen und Probleme im Zusammenhang mit der `null`-Referenz:

```

Color a,b;
a = new Color(255,0,255);
b = null;          // zulaessig

if (a != b) {    // Vergleich ok
    String s = b.toString(); // NullPointerException
}

```

2.2.3 Lebensdauer von Objekten und Instanzvariablen

In Java lassen sich Objekte nicht löschen. Aus Sicht des Programmierers leben Objekte von der Erzeugung bis zum Ende der Ausführung des Programms.

Der Speicher nicht erreichbarer Objekte wird ggf. vor Ablauf der Lebensdauer von der automatischen Speicherbereinigung frei gegeben (\Rightarrow Abschnitt "Speicherverwaltung").

2.3 Methodenaufruf (engl. method invocation)

Syntax:

Ein Methodenaufruf ist ein Ausdruck ähnlich einem Prozeduraufruf, allerdings mit einem zusätzlichen Parameter:

[Ausdruck](#) →

[Ausdruck](#) . << *Bezeichner* >> ([AktuelleParameterListe](#))

Semantik:

1. Werte den Ausdruck vor dem Punkt aus; dieser muss eine Referenz liefern.
Liefert dieser `null`, löse eine `NullPointerException` aus. Andernfalls liefert er die Referenz auf ein Objekt X .
2. Werte die aktuellen Parameter p_1, \dots, p_n aus.
3. Führe den Rumpf der zu X gehörenden, angegebenen Methode mit
 - X als implizitem Parameter (`this`) und
 - p_1, \dots, p_n als expliziten Parametern aus.

Das Ergebnis des Aufrufs ist der Rückgabewert der Ausführung des entsprechenden Methodenrumpfes.

Unterscheidung zwischen Instanzmethoden und statischen Methoden

Wir kennen nun zwei Arten von Methoden in Java:

- Instanzmethoden, die Datentypoperationen für die Instanzen bzw. Objekte bereitstellen und es erlauben mit den Objekten zu interagieren
- Statische Methoden, die Prozeduren implementieren und nicht an Objekte gebunden sind

Das folgende Beispiel zeigt die Verwendung beider Arten von Methoden im Vergleich:

```
public class Luminance {
    // Berechnet die Luminanz, d.h. die effektive Helligkeit einer Farbe
    public static double lum(Color color) {
        int r = color.getRed();
        int g = color.getGreen();
        int b = color.getBlue();
        return 0.299 * r + 0.587 * g + 0.114 * b;
    }
}

public class LuminanceTest {
    public static void main (String[] args) {
        int r = Integer.parseInt(args[0]);
        int g = Integer.parseInt(args[1]);
        int b = Integer.parseInt(args[2]);
        Color c = new Color(r,g,b);

        double luminance = Luminance.lum(c);
        double luminanceBright = Luminance.lum(c.brighter());
        ...
    }
}
```

Die folgende Übersicht ist eine Zusammenfassung der Unterschiede bei der Verwendung:

	Instanzmethoden	statische Methoden
Beispielaufruf: Aufgerufen mit Parameter	<code>c.getGreen()</code> Objektreferenz Referenz auf Objekt und Argument(e)	<code>Math.sqrt(2.0)</code> Klassenname Argument(e)