Kapitel 09:
Einführung in die Objektorientierung

Software Entwicklung 1

Annette Bieniusa, Mathias Weber, Peter Zeller

> "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.

Lernziele dieses Kapitels:

- Objektcharakteristiken zu kennen und an Objekten zu benennen.
- Abstraktion durch Schnittstellenbildung zu erläutern.
- In Java Objekte zu erzeugen, zu referenzieren und mit ihnen zu interagieren.
- Zwischen Operationen auf Referenzen von Operationen auf Objekten zu unterscheiden.
- Instanzmethoden und statischen Methoden zu unterscheiden.

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.

Aufenthaltsort 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:Auto Wir erläutern die Objektcharakteristiken am Beispiel eines Autos:

- Typische Attribute und ihr Zustand für ein konkretes Auto:

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

- Die Lebensdauer dieses Autos erstreckt sich von der Konstruktion, die am 03. Juli 1996 abgeschlossen war, bis zur Verschrottung, die allerdings noch aussteht.
- Der aktuelle Ort ist der Parkplatz vor Gebäude 32 an der TU Kaiserslautern.
- Verhalten, d.h. Nachrichten, die von dem Auto verstanden werden:
  - → bremsen
  - → Gas geben
  - → Scheibenwischer (an/aus)
  - → ...
- Das Auto hat auch eine Identität; wir können es von anderen Autos gleichen Typs, gleicher Farbe, gleichen Baujahrs etc. unterscheiden.

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

Im Gegensatz zu den bisher betrachteten Objekten haben Zahlen

- keinen (veränderlichen) Zustand,

- keine Lebensdauer,
- keinen Aufenthaltsort,
- kein Verhalten,
- keine Identität jenseits der Gleichheit.

Deshalb nennen wir sie *Werte*.[1]

1.1 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 bzw. interagieren.

1.2 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.
- Frau B. besitzt eine Methode, wie mit dem Auftrag zu verfahren ist. (Herr K. weiß nicht, wie Frau B. den Auftrag ausführt.)
- Nach Ausführung schickt sie Herrn K. das Buch zu.

[1] Allerdings können Werte durch Objekte repräsentiert werden. Dies wird in mehreren Programmiersprachen so umgesetzt. Diese sind dann i.d.R. unveränderlich.

1.3 Objektbasiertes System

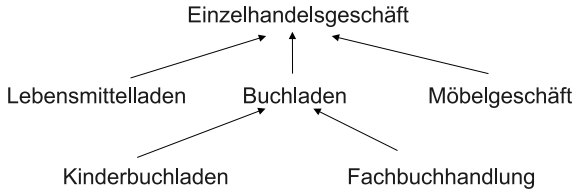
In der Objektorientierung werden Systeme als Menge kooperierender Objekte modelliert, die untereinander in Beziehung stehen und über Nachrichten miteinander kooperieren. Derartige Systeme heißen ***objektbasiert***.
Grundsätzlich gibt es zwei Konzepte zur programmiersprachlichen Beschreibung von Objekten:

- ***Prototyp-Konzept:***
  Der Programmierer beschreibt direkt einzelne Objekte. Neue Objekte werden durch Klonen bzw. Vervielfältigen 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 neue Objekte als Instanzen der Klassen zu erzeugen. Typischerweise wird die Klasse dabei aber nicht verändert.

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: Einzelhandelsgeschäfte Einzelhandelsgeschäfte 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, aber auch 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***.

1.4 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

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

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 auf der Objekterzeugung und Methodenaufruf. Der Entwurf und die Implementierung eigener Datentypen behandeln wir im nächsten Kapitel.

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 u.a. in Form von Klassen implementiert. Der Klassenname wird dabei auch 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 `Color`-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; der Wert 255 repräsentiert die stärkste Intensität. Weitere Farbtöne können durch additive Farbmischung dargestellt werden.

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 java.awt.Color	
	Color(int r, int g, int b)	Konstruktor
int	getRed()	Intensität für Rot
int	getGreen()	Intensität für Grün
int	getBlue()	Intensität für Blau
Color	brighter()	Hellere Version dieser Farbe
Color	darker()	Dunklere Version dieser Farbe
String	toString()	String-Darstellung dieser Farbe
boolean	equals(Color c)	Vergleich[2]

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. Die Methoden `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.

[2] Genau 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.

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. Da diese Bibliothek in der Standard-Installation von Java enthalten ist, ist eine zusätzliche .jar-Datei o.ä. nicht notwendig.

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

```
1  import java.awt.Color;
2
3  public class ColorSquares {
4    public static void main(String[] args) {
5      int r = Integer.parseInt(args[0]);
6      int g = Integer.parseInt(args[1]);
7      int b = Integer.parseInt(args[2]);
8
9      Color c1 = new Color(r,g,b);
10     Color c2 = c1.brighter();
11     Color c3 = c1.darker();
12
13     StdDraw.setPenColor(c3);
14     StdDraw.filledSquare(.25,.5,.3);
15
16     StdDraw.setPenColor(c1);
17     StdDraw.filledSquare(.25,.5,.2);
18
19     StdDraw.setPenColor(c2);
20     StdDraw.filledSquare(.25,.5,.1);
21
22   }
23 }
```

- In Zeile 9 wird ein `Color`-Objekt mit RGB-Werten erzeugt (siehe Abschnitt2.2). Eine Referenz auf dieses Objekt wird der Variablen `c1` zugewiesen.
- Durch den Methodenaufruf `c1.brighter()` wird eine Referenz auf ein weiteres `Color`-Objekt mit hellerem Farbton geliefert und der Variablen `c2` zugewiesen; ähnlich für `c3`.
- Mit Hilfe der Bibliothek `StdDraw` (siehe Vorlesungshomepage) werden nun Quadrate in den drei Farben gezeichnet. Dabei wird zunächst der aktuelle Farbton festgelegt (Zeile 13). Die Prozedur `StdDraw.filledSquare(double x, double y, double radius)` zeichnet dann ein Quadrat mit Zentrum (x, y) mit Seitenbreite $2 * \text{radius}$.

Frage 2: Führen Sie das Programm mit verschiedenen Parameterwerten aus!

Beispiel: API von String Auch Strings sind in Java Objekte. Sie sind in der Standardbibliothek eingebaut und haben in Java eine eigene spezielle Syntax, die wir schon gesehen haben (`"Hello World"`). Hier ein Auszug aus der API von `String`:

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

- 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
```

Zum Vergleich von Objekten siehe Abschnitt 2.2.1.

2.2 Objekte erzeugen und referenzieren

Um neue Objekte zu erzeugen, werden Java ***Konstruktoren*** verwendet. Bei der Objekterzeugung kann oft mit Hilfe von Parametern der Initialzustand eines Objekts

parametrisiert werden. So wird der Farbton eines `Color`-Objects mit Hilfe der Parameter `r`,`g`,`b` bei der Erzeugung des Objekts festgelegt. Die API der String-Klasse hat zwei Konstruktoren mit unterschiedlichen Parametern. Alle Konstruktoren einer Klasse haben dabei den gleichen Namen wie die Klasse selbst.

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

Ausdruck →
`new` ≪ *Bezeichner* ≫ `(`AktuelleParameterListe`)`

wobei

AktuelleParameterListe → ε
| AusdruckListe

AusdruckListe → Ausdruck
| Ausdruck , AusdruckListe

Semantik:

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

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.
- Eine Variablen von einem Referenztypen speichern nicht das Objekte als Ganzes, sondern lediglich eine Referenz auf das Objekt.

Beispiel Von einer Klasse lassen sich mit Hilfe der Konstruktoren 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 Farbwert repräsentieren. `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, da `c` und `e` den gleichen Farbton repräsentieren.

2.2.2 Die null-Referenz

Wie wir im Kapitel 05 bei den 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

In Java lassen sich Objekte nicht löschen. Aus Sicht des Programmierers leben Objekte von ihrer 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 (näheres dazu in einem späteren Kapitel zu “Speicherverwaltung”).

2.3 Methodenaufruf (engl. method invocation)

Eine Methode implementiert die Handlungsvorschrift eines Dienstes, das durch ein Objekt angeboten wird.
In Syntax und Semantik ähnelt ein Methodenaufruf dem Aufruf einer Prozedur, allerdings mit dem Zielobjekt als einem zusätzlichen, impliziten Parameter.
Syntax:

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 $p_1, \ldots, p_n$ aus.
3. Führe den Rumpf der zu X gehörenden, angegebenen Methode mit
   - X als implizitem Parameter (`this`) und
   - $p_1, \ldots, p_n$ 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***, welche die Operationen eines Datentyps 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:

```
import java.awt.Color;

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 = 100;
    int g = 0;
    int b = 150;
    Color c = new Color(r,g,b);

    double luminance = Luminance.lum(c);
    double luminanceBright = Luminance.lum(c.brighter());
    StdOut.println(luminance + " vs " + luminanceBright);
   }
  }
```

Die statische Methode `lum`, die in der Klasse `Luminance` definiert wird, kann durch `Luminance.lum(c)` aufgerufen werden. Dazu wird kein Objekt der Klasse `Luminance` erzeugt oder benötigt. Die Instanzmethode `brighter` kann hingegen nur auf `Color`-Objekten aufgerufen werden; d.h. wir benötigen eine Referenz auf ein `Color`-Objekt, um die Methode zu verwenden.

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

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

In diesem Kapitel stand die Erzeugung und Interaktion mit Objekten im Vordergrund. Wie wir Klassendeklarationen für eigene Referenztypen und Objekte definieren können, werden wir im nächsten Kapitel behandeln.

Hinweise zu den Fragen

Hinweise zu Frage 1: Physische Objekte sind beispielsweise:

- Stühle, Tische, DVD-Spieler, Computer,
- Autos, Staudämme,
- Häuser, Städte, Länder, Flüsse, Atmosphäre,
- Personen, Tiere, Pflanzen, Augen, Organe,
- Bücher, Bibliotheken, Läden

Auch immaterielle Konstrukte und Werte haben Objektcharakter:

- Vorlesungen, Prüfungen
- Reisen, Ruderregatten, Kriege
- Gesetze, Regeln, Pläne
- Konten, Börsenkurse, Optionsscheine
- Sprachen, Völker
- Web-Server, Telefonnetze, GUI-Fenster

Begriffe ohne Objektcharakter sind hingegen folgende:

- Wachstum, Größe
- Klugheit, Liebe, Schönheit, Sein
- Demokratie, Humor
- Einigkeit, Gerechtigkeit, Freiheit
- Freizeit, Arbeit
- Effizienz, Information, Sichtbarkeit