TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa

AG Softech
FB Informatik
TU Kaiserslautern

Lernziele

- Idee der Objektorientierung verstehen
- Objektcharakteristiken kennen und an Objekten erläutern
- Abstraktion durch Schnittstellenbildung erläutern
- Objekte erzeugen, referenzieren und mit ihnen interagieren in Java
- Operationen auf Referenzen von Operationen auf Objekten unterscheiden
- Unterschied zwischen Instanzmethode und statischer Methode kennen

Einführung in die Objektorientierung

Grundlegende Idee

"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."

aus: O. Lehrmann Madsen, B. Møller-Petersen, K. Nygaard: Object-oriented Programming in the BETA Programming Language, Addison-Wesley, 1993.

Objektorientiertes Paradigma

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**.

Zum Objektbegriff: 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.

Zum Objektbegriff: Lebensdauer

- Jedes Objekt existiert von seiner Entstehung bis zum Verschwinden bzw. von seiner Erzeugung bis zur Löschung.
- Die Lebensdauer kann in Zeit oder in Ablaufschritten gemessen werden.

Zum Objektbegriff: Aufenthaltsort

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

Zum Objektbegriff: 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 **Methode**, die beschreibt, wie eine Nachricht bearbeitet wird.

Zum Objektbegriff: Identität

- Zu einem Objekt gibt es im Allg. gleichartige Objekte (z.B. zwei Bücher, zwei Häuser,...)
- Objekte kann man vergleichen.
- Objekte 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

- 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: A1 zwischen Ennepetal und Volmarstein
- Verhalten, d.h. Nachrichten, die verstanden werden:
  - → bremsen
  - → Gas geben
  - → Scheibenwischer (an/aus)
  - → ...
- Identität: gegeben.

Beispiele: Dinge mit Objektcharakter

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

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

Bemerkung

Die üblichen mathematischen "Objekte" (Funktionen, Mengen, Zahlen) 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.)

Beziehungen zwischen Objekten

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

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

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 & Aufträge

Modellierung eines Buchkaufs:

- Buchhändlerin B bietet den Dienst an, per Email 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.

Begriffsklärung: 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**.

Konzept: Prototyp vs. Klasse

Grundsätzlich gibt es zwei Konzepte zur programmiersprachlichen Beschreibung von Objekten:

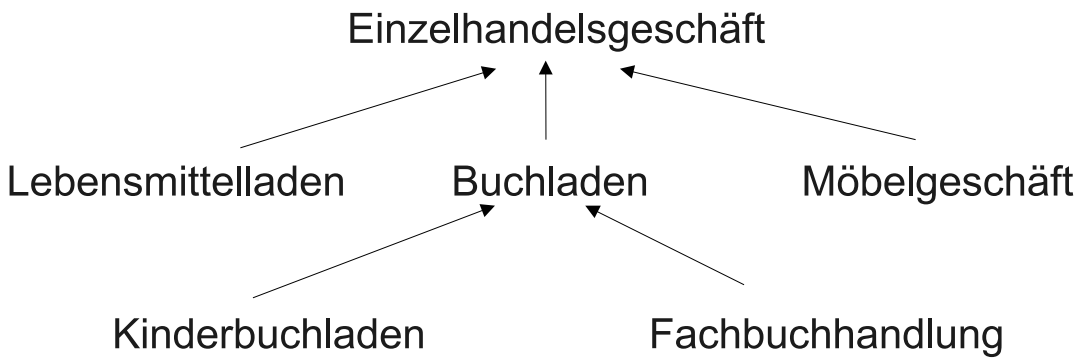
- **Prototyp-Konzept:**
  Der Programmierer beschreibt direkt einzelne Objekte. Neue Objekte werden durch Klonen existierender Objekte und Verändern ihrer Eigenschaften zur Laufzeit erzeugt.
- **Klassenkonzept:**
  Der Programmierer deklariert Klassen als Beschreibung der Eigenschaften, die Objekte dieser Klasse haben sollen. Die Programmiersprache ermöglicht es, zur Laufzeit Objekte der Klassen zu erzeugen, aber nicht, die Klassen zu verändern.

Wir betrachten hier nur das Klassenkonzept.

Klassifikation

Objekte lassen sich nach ihren Eigenschaften klassifizieren:

- Alle Objekte mit ähnlichen Eigenschaften werden zu einer Klasse zusammengefasst.
- Die Klassen werden hierarchisch geordnet.
- Die übergeordneten Klassen besitzen Eigenschaften, die allen untergeordneten Klassen gemeinsam sind.



Beobachtung

- Unterscheide: Objekt $\leftrightarrow$ Klasse
- 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.

Begriffsklärung: Objektorientiertes System

Objektbasierte Systeme, bei denen

- die Objekte Klassen zugeordnet und
- die Klassen gemäß einer Klassifikation hierarchisch geordnet sind und Vererbung erlauben,

heißen **objektorientiert**.

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.

Objektorientierte Programmierung in Java

Java ist objektorientiert

- Objekte und Klassen sind native Abstraktionsmechanismen in Java
- Objekte nehmen Werte von **Referenztypen** an (z.B. Klassentypen)
- Referenztypen sind definiert in Bibliotheken bzw. vom Programmierer
- *In dieser Vorlesung:* Verwendung von Objekten
  - Objekterzeugung
  - Attributzugriff
  - Methodenaufruf
  - Objektlöschung (in Java nicht direkt unterstützt)
- (Definition von eigenen Referenztypen $\Rightarrow$ nächste Vorlesung)

Abstraktion durch Schnittstellenbildung

Um einen Datentypen zu verwenden, muss man nicht wissen, wie er implementiert ist.

API von Referenztypen

Auszug aus der API für `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

Verwendung von `Color`-Objekten

```
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();
    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);

  }
}
```

Objekte erzeugen

Syntax in Java:

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

wobei

AktuelleParameterListe → ε
| AusdruckListe

AusdruckListe → Ausdruck
| Ausdruck , AusdruckListe

Semantik:

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

Ergebnis ist die Referenz des neu erzeugten Objekts.

Begriffsklärung: Referenz, Verweis, Zeiger

Eine **Objektreferenz** (engl. *object reference*) ist eine eindeutige abstrakte Adresse oder Bezeichnung für ein Objekt.

Variablen speichern nicht die Objekte als Ganzes, sondern Objektreferenzen.

Die Auswertung von Ausdrücken eines Klassentyps K liefert Referenzen auf Objekte des Typs K.

Operationen auf Referenzen

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

- Test auf Gleichheit mit == bzw. mit != auf Ungleichheit.
  Zwei Referenzen sind genau dann gleich, wenn sie dasselbe Objekt referenzieren.

Wozu wertet `c == d` und `c == e` und `d == f` aus?

Operationen auf Referenzen

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

- Test auf Gleichheit mit == bzw. mit != auf Ungleichheit.
  Zwei Referenzen sind genau dann gleich, wenn sie dasselbe Objekt referenzieren.

Wozu wertet `c == d` und `c == e` und `d == f` aus?

- Um **Objekte** auf Gleichheit zu testen, verwenden wir die `equals()`-Methode.

Wozu wertet `c.equals(e)` und `d.equals(f)` aus?

Die `null`-Referenz

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

if (a != b) {  // Vergleich ok
  String s = b.toString();  // Was passiert hier?
}
```

Lebensdauer von Objekten

In Java lassen sich Objekte nicht löschen.

Aus Sicht des Programmierers leben *Objekte* und deren *Instanzvariablen* von der Objekterzeugung bis zum Ende der Ausführung des Programms.

Der Speicher nicht erreichbarer Objekte wird ggf. vor Ablauf der Lebensdauer von der automatischen Speicherbereinigung frei gegeben (siehe späteres Kapitel zu "Speicherbereinigung").

Methodenaufruf (engl. method invocation) I

Syntax:

Ausdruck $\rightarrow$
Ausdruck . $\ll Bezeichner \gg$ (AktuelleParameterListe)

Semantik:

- Werte den Ausdruck aus; dieser muss eine Referenz liefern.
- Liefert dieser `null`, löse eine `NullPointerException` aus.
- Andernfalls liefert er die Referenz auf ein Objekt X. Werte die aktuellen Parameter $p_1, \ldots, p_n$ aus.
- Führe den Rumpf der angegebenen Methode mit
  - X als implizitem Parameter und
  - $p_1, \ldots, p_n$ als expliziten Parametern aus.

Methodenaufruf (engl. method invocation) II

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

Beispiele:

```
Color c1 = new Color(r,g,b);

int i    = c1.getRed();
String s = c1.toString();
Color c2 = c1.brighter();
```

Instanzmethoden vs. statischen Methoden

```
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) {
  ...
  Color c = new Color(r,g,b);

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

Überblick: Instanzmethoden vs. statische Methoden

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

- Statische Methoden "gehören" zu den Klassen, nicht zu den Objekten!
- Um eine statische Methode aufzurufen braucht man keine Referenz auf ein Objekt.
- In dieser Vorlesung: Statische Methoden als Prozeduren!