TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa / Arnd Poetzsch-Heffter

AG Softech
FB Informatik
TU Kaiserslautern

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Einführung in die Objektorientierung

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

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.

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

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.

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Beispiele: Physische Objekte

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

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Beispiele: Gedankliche Objekte und Werte

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

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Beispiele: Begriffe ohne Objektcharakter

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

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

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 hat eine Lebensdauer; das 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.

Zum Objektbegriff: Aufenthaltsort

- Objekte besitzen normalerweise einen Ort, der durch eine Adresse charakterisiert wird.
- Beispiele für Adressen: Hausadresse, Emailadresse, Web-Adresse, Telefonnummer, Geo-Koordinaten, Rechneradresse, Adresse 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 sogenannte 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.

Bemerkungen

- Die *charakteristischen Eigenschaften* Zustand, Lebensdauer, Ort, Verhalten und Identität sind als Hilfestellung, nicht als scharfe Begriffsklärung zu verstehen.
- Anhand der charakteristischen Eigenschaften lässt sich untersuchen, wie ausgeprägt der Objektcharakter eines Gegenstands ist.

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.

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

Objekte können zueinander in unterschiedlichen Beziehungen stehen:

- 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.

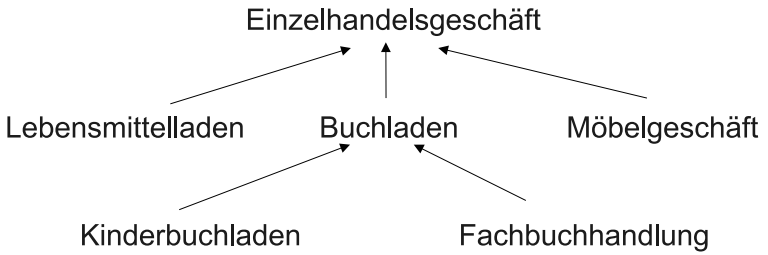
Klassifikation und Vererbung

Objekte lassen sich nach ihren Eigenschaften klassifizieren:

- Alle Objekte mit ähnlichen Eigenschaften werden zu einer Klasse zusammen gefasst.
- Die Klassen werden hierarchisch geordnet.

Beispiel: Klassifikation

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



- Die übergeordneten Klassen besitzen Eigenschaften, die allen untergeordneten Klassen gemeinsam sind.
- Beachte die Unterscheidung: Objekt $\leftrightarrow$ Klasse

Beobachtung

- 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.

Vorteile der Klassifikation

- 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.

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.

Die Menge der möglichen Zustände eines Objekts entspricht den Wertebereichen seiner Attribute.

Die Reaktionen eines Objekts auf eintreffende Nachrichten legen sein dynamisches Verhalten fest.

Objekte und Klassen in Java

Klassen in Java I

Eine einfache **Klassendeklaration** in Java hat folgende Bestandteile:

```
class Person {                          Klassenname
  String name;                          Attribut
  Person(String n) {                    Konstruktor
    this.name = n;
  }
  String getName() {                    Methode
    return this.name;
  }
}
```

Ein Java-Objekt kann genau auf die Nachrichten reagieren, für die Methoden in seiner Klasse deklariert sind oder für die es Methoden geerbt hat (siehe Abschnitt “Vererbung”).

Klassen in Java II

- In der Regel speichern wir den Code für die Deklaration einer Klasse in einer gleichnamigen `.java` - Datei ab.
  Im Beispiel: `Person.java`
- Klassennamen beginnen in Java nach Konvention mit einem Großbuchstaben.

Deklaration von Klassen: Klassenname

Direkt hinter dem Schlüsselwort `class` wird der Name der Klasse angegeben. 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:

```
Person m(Person p) { ...  Person vater;  ... }
```

Außer den Objekten einer Klasse K gehören auch alle Objekte von Unterklassen von K zum Typ K (siehe Abschnitt "Vererbung").

Deklaration von Klassen: Attribute I

Innerhalb einer Klasse K können Attribute deklariert werden.

Für jedes in K deklarierte Attribut vom Typ T besitzen die Objekte der Klasse K eine **objektlokale** Variable vom Typ T. Diese objektlokalen Variablen nennt man häufig auch **Instanzvariablen**.

Deklaration von Klassen: Attribute II

Die Lebensdauer der Instanzvariablen entspricht der Lebensdauer des Objekts.

Klasse:

Person
name : String
getName() : String

Objekt:

: Person
name =

Die Objekte einer Klasse K nennt man auch **Instanzen** oder *Ausprägungen* von K.

Deklaration von Klassen: Methoden I

Innerhalb der Klassendeklaration können beliebig viele *Methoden* deklariert werden.

Methodendeklarationen bestehen aus einer Signatur und einem Methodenrumpf. Syntaktisch sind sie wie Prozedurdeklarationen aufgebaut.

Außer den deklarierten Parametern besitzt jede Methode *m* einen weiteren, sogenannten **impliziten Parameter** vom Typ der Klasse, in der *m* deklariert wurde. Dieser Parameter wird im Methodenrumpf mit `this` bezeichnet.

Deklaration von Klassen: Methoden II

Beispiel:

```
String getName() {
  return this.name;
}
```

Neben den prozeduralen Anweisungen kann eine Methodenrumpf in Java:

- neue Objekte erzeugen,
- auf Attribute zugreifen,
- Nachrichten an andere Objekte schicken (Methodenaufruf).

Konstruktoren

Konstruktoren erzeugen und initialisieren Objekte.

Sie haben den gleichen Namen wie die Klasse, in der sie deklariert sind.

Beim Start der Ausführung eines Konstruktors ist das zugehörige Objekt bereits erzeugt, seine Attribute jedoch nur mit Standardwerten initialisiert.

Konstruktoren liefern als Ergebnis das neu erzeugte Objekt zurück, genauer: eine Referenz auf dieses Objekt.

Initialisierung I

Attribute (wie auch lokale Variablen) können direkt an ihrer Deklarationsstelle initialisiert werden.

```
class C {
  int a = 78;
  C() {}
}
```

```
class C {
  int a;
  C() {
    a = 78;
  }
}
```

Die Initialisierung von Attributen erfolgt vor dem Eintritt in den Konstruktor*rumpf*.

Initialisierung II

In Java können Attribute und Variablen durch das Schlüsselwort `final` als **unveränderlich** deklariert werden.

In diesem Fall *muss* die Initialisierung an der Deklarationsstelle erfolgen oder in geeigneter Weise im Konstruktor.

```
class Mathe {
    ...
    final float PI = 3.141; // Konstante
    ...
}
```


Benutzung von Objekten

Benutzung von Objekten

Benutzung von Objekte:

- Objekterzeugung
- Attributzugriff
- Methodenaufruf
- Objektlöschung (in Java nicht direkt unterstützt)

Objekte: Erzeugen und Referenzieren

Syntax in Java:

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

```
new <Konstruktorname> ( <Parameterliste> )
```

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. Manchmal spricht man auch von **Verweis** (engl. *link*) oder **Zeiger** (engl. *pointer*).

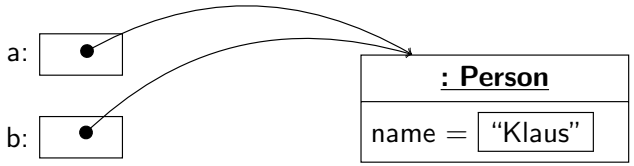
Variablen speichern nicht die Objekte als Ganzes, sondern Objektreferenzen.

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

Beispiel: Objektreferenz

Folgendes Programmfragment verdeutlicht die Unterscheidung zwischen Objekt und Referenzen:

```
Person a,b;
a = new Person ("Klaus");
b = a;     // a und b speichern dieselbe Objektreferenz
b.getName();  // liefert "Klaus"
```



Es gilt also: `a.name == b.name`

Sprechweisen & Bemerkungen

- Häufig spricht man von Referenzen auf ein Objekt.
- Referenzen nennt man auch anonyme Namen oder Bezeichner. Eine Referenz ist ein Wert.
- Referenzen stellt man graphisch üblicherweise durch Pfeile dar. Zwei Referenzen sind gleich, wenn sie auf das gleiche Objekt zeigen.
- Die Unterscheidung Referenz / Objekt hat viele Analogien:
  - Anschrift / Wohnung
  - Email-Adresse / Mailbox
  - Telefonnummer / Telefonanschluss
  - Speicheradresse / Speicherzelle

Beispiel: Referenzsemantik

```
class Uebungsgruppe {
  int anzahl;
  Uebungsgruppe (int anzahl) {
    this.anzahl = anzahl;
  }
}
```

Verfolgen Sie schrittweise das folgende Programm im Java-Visualizer:

```
Uebungsgruppe a, b, c;
a = new Uebungsgruppe(30);
b = a;
c = new Uebungsgruppe(30);
if (a != c) {
  StdOut.println("Es gibt doch zwei Gruppen!");
}
```

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 `c` gilt:
  `a == b` und `a != c` und `b != c`

- Über Referenzen kann man Objekten Nachrichten schicken und auf sie zugreifen, d.h. auf ihre Instanzvariablen.

Bemerkungen

- Objekte können der gleichen Klasse angehören und den gleichen Zustand haben (gleich sein), aber trotzdem nicht die dieselbe Identität haben und damit auch unterschiedliche Referenzen besitzen.
- Variablen von einem Klassentyp speichern Referenzen. Wir sagen deshalb auch vereinfachend, dass eine Variable ein Objekt referenziert.
- Es ist wichtig zwischen einer Variablen und dem Objekt, das sie referenziert, zu unterscheiden!

Die `null`-Referenz

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

```
Person a,b;
a = new Person ("Klaus");
b = null;       // zulaessig

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

Ebenso würde der Methodenaufruf `b.getName()` eine `NullPointerException` liefern.

Begriffsklärung: Objektgeflecht

Eine Menge von Objekten, die sich gegenseitig referenzieren, nennen wir ein **Objektgeflecht**.

Bemerkungen:

- Objektgeflechte werden zur Laufzeit aufgebaut und verändert, sind also dynamische Entitäten.
- Klassendiagramme kann man als vereinfachte statische Approximationen von Objektgeflechten verstehen.

Lebensdauer von Objekten und Instanzvariablen

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 (vgl. Abschnitt "Speicherbereinigung").

Attributzugriff I

Syntax in Java:

Auf Instanzvariablen von Objekten kann mit Ausdrücken folgender Form zugegriffen werden:

```
<Ausdruck> . <Attributbezeichner>
```

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; in dem Fall liefert der gesamte Ausdruck die Instanzvariable von X zum angegenen Attribut (L-Wert) oder deren Wert (R-Wert).

Attributzugriff II

Abkürzende Notation:

Der implizite Methodenparameter `this` kann beim Zugriff auf ein Attribut `a` weggelassen werden, d.h.

```
a
```

ist gleichbedeutend mit

```
this.a
```

innerhalb von Klassen, in denen `a` deklariert ist.

Beispiel: Attributzugriffe

```
class Autor {
  String name;
  int    geburtsjahr;
}


class Buch {
  Autor  autor;
  String titel;

   void printInfo() {
      StdOut.println("Titel:" + this.titel);
      StdOut.println("Autor:" + this.autor.name);
   }
}
```

Methodenaufruf (engl. method invocation) I

Syntax:

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

```
<Ausdruck> . <Methodenbezeichner> (<AktParamliste>)
```


Methodenaufruf (engl. method invocation) II

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.

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

Bemerkung

Abkürzende Notation:

Wie beim Attributzugriff kann auch beim Methodenaufruf der implizite Methodenparameter `this` weggelassen werden, also `m(...)` statt `this.m(...)`.

Beispiel: Methodenaufrufe

```
class Mensch {
  Mensch vater, mutter;
  String name;

  Mensch getOpa (boolean mutterseits) {
    if (mutterseits) {
      return mutter.vater;
    } else {
      return vater.vater;
    }
  }

  void ermittleOpa(Mensch m) {
     Mensch opaV;
     String opaMName;
     opaV     = m.getOpa(false);
     opaMName = m.getOpa(true).name;
  }
}
```

Objektorientierte Programme

Ein objektorientiertes Java-Programm Π besteht aus einer Menge von Klassen. Mindestens eine der Klassen muss eine Methode mit Namen `main` und folgender Signatur besitzen:

```
public static void main (String[] args) {   ...   }
```

Beim Start von Π wird die Klasse angegeben, deren main-Methode ausgeführt werden soll:

```
java  <Klassenname>  <arg1>  <arg2>  ...
```

Die Argumente `arg1`,... werden dabei der main-Methode im Parameter `args` als ein Feld von Strings übergeben.

Bei der Ausführung werden die benötigten Objekte erzeugt. Diese Bearbeiten ihre Aufträge durch Ausführung von Methoden.