Softwareentwicklung II (SE2):

Objektorientierte Programmierung und Modellierung (OOPM)

Vorlesung 11 – Fachwerte und Werttypen

Matthias Riebisch, Detlef Litterst

Unter Verwendung von Materialen von Axel Schmolitzky,
Heinz Züllighoven und Guido Gryczan

SE1 und SE2: Als Tutor mitwirken!

- Unterstützen,
- Verstehen helfen,
- Besser über Software diskutieren,
- Positive Rückmeldung bekommen,
- SE1 der Zukunft mitgestalten,
- Bezahlung über Tutor-Vertrag

Ihr möchtet dabei sein? Mail an
Volodymyr Biryuk
<biryuk@informatik.uni-hamburg.de>

SE1 und SE2 lebt von engagierten Tutoren!

Vorlesung 11 – Fachwerte und Werttypen

Fragen zur Prüfungsvorbereitung

1. Erläutern Sie, warum Softwaresysteme besser änderbar sind, wenn sie ähnlich der realen Welt aufgebaut sind!
2. Welche Eigenschaften von Fachwerten passen schlecht zu einer Realisierung als Objekte?
3. Vergleichen Sie die Konzepte *Wert* und *Objekt*!
4. Stellen Sie die verschiedenen Definitionen von Gleichheit von Objekten gegenüber!
5. Erläutern Sie das Typesafe Enum Pattern!
6. Geben Sie am Beispiel einer Klasse für Postleitzahlen an, welche 6 Richtlinien bei der Implementierung von Fachwerten als Klassen eingehalten werden müssen!

Wiederholg. VL07: Fachwerte

DEF

- beschreiben wertartige Dinge, wie **Kontonummer** oder **Geldbetrag**
- fachlich motivierte **Werte** – auch allgemeiner wie Zahlen und Zeichenketten
  - Ein Wert ist **unveränderlich**
  - programmiertechnisch durch **Werttypen** realisiert
  - Werttypen sind besondere Typen mit einer **unveränderlichen Wertemenge**; Werte werden somit konzeptuell nicht erzeugt, sondern bei Bedarf aus der Wertemenge ausgewählt.
- bilden die grundlegenden Typen in einem Anwendungssystem.

Motivation

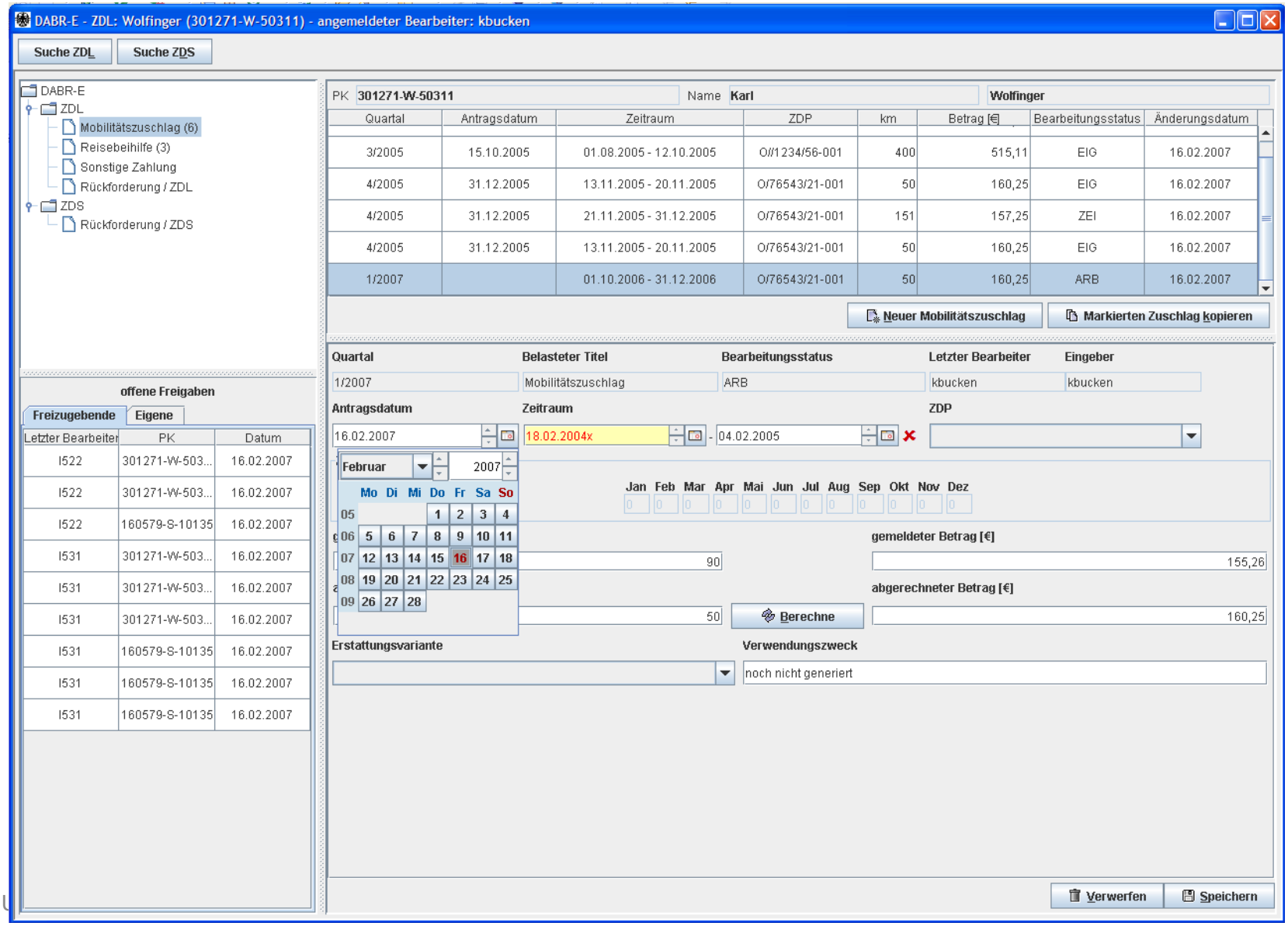
Softwaresysteme als Abbildung der realen Welt → Änderungen vereinfacht

- Änderungen von realen (oder imaginären) Elementen der Welt lassen sich leichter auf Änderungen des Softwaresystems umsetzen
  - Erleichtertes Feature Localization
    (Was muss geändert werden)
  - Erleichterter Entwurf von Änderungen anhand der Änderung des Problemraums (Welche Änderungen müssen vollzogen werden)

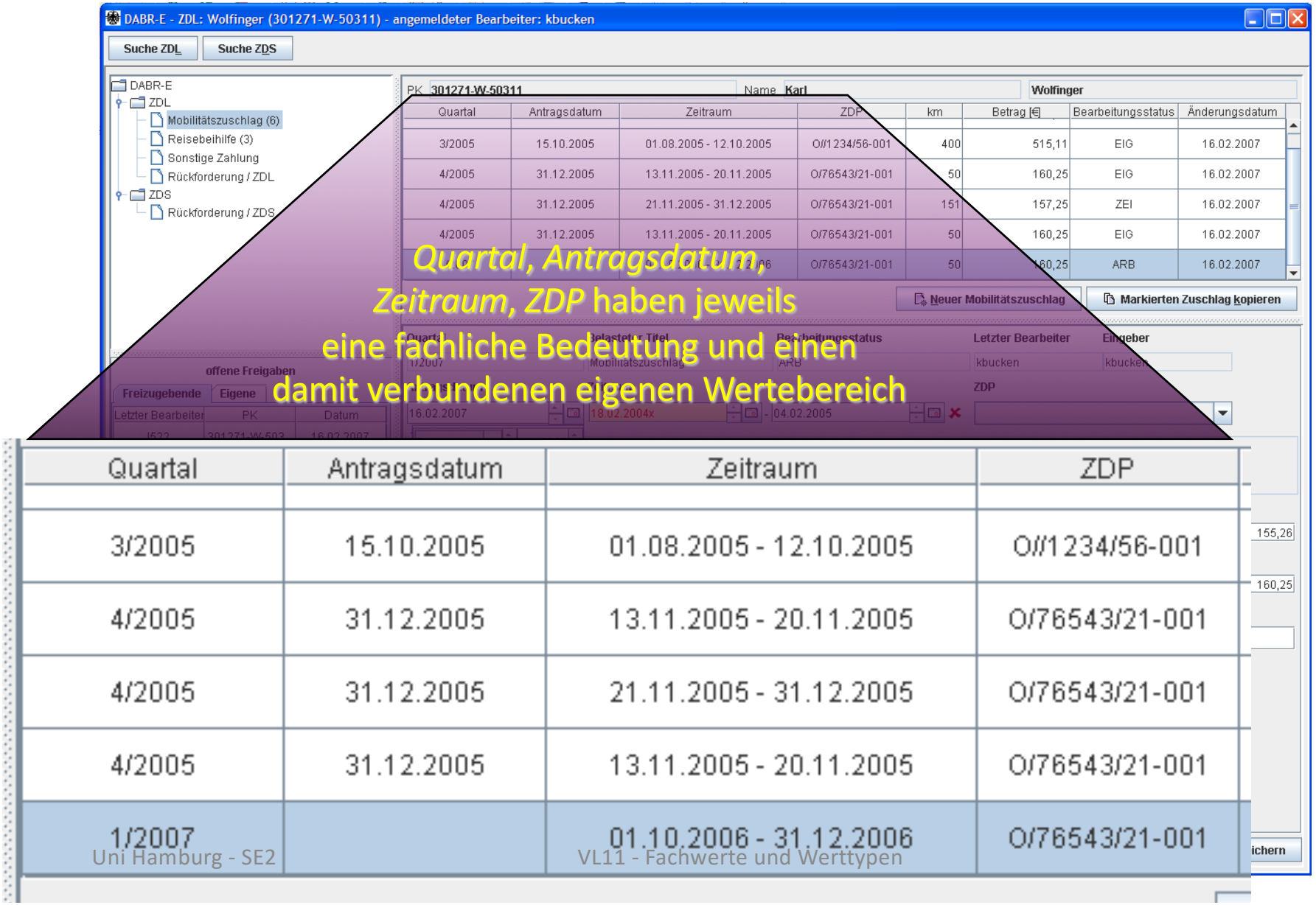
Objektorientierung: Elemente der realen Welt als Objekte mit Struktur und Verhalten in Software abbilden

WAM: Interaktionen zwischen Mensch und Softwaresystem auf Bearbeitung von Material mit Werkzeugen abbilden

Beispiel: Auftragswerkzeug zur Bearbeitung von Zahlungsaufträgen



Materialien enthalten fachlich motivierte Werte



Quartal	Antragsdatum	Zeitraum	ZDP
3/2005	15.10.2005	01.08.2005 - 12.10.2005	O//1234/56-001
4/2005	31.12.2005	13.11.2005 - 20.11.2005	O/76543/21-001
4/2005	31.12.2005	21.11.2005 - 31.12.2005	O/76543/21-001
4/2005	31.12.2005	13.11.2005 - 20.11.2005	O/76543/21-001
1/2007		01.10.2006 - 31.12.2006	O/76543/21-001

Fachlich motivierte Werte in jedem Anwendungsbereich erkennbar

Zeitpunkt	Zeitpunkte werden nicht erzeugt
Zeitraum einer Klausur	Zeitraum für Klausur „ändern“ bedeutet eigentlich nur einen anderen Zeitraum „wählen„
Geldbetrag	„100 Euro“ sind sogar in doppelter Hinsicht ein Wert
Kontonummer eines Bankkunden	Bei einem Bankkunden die Kontonummer „ändern“ bedeutet lediglich Ändern der Zuordnung zwischen Kunde und Nummer – nicht der Nummer selbst
Sozialversicherungsn…ner	Erhält ein neuer Mitbürger eine SVN, dann wird diese nicht frisch erzeugt, sondern es wird ihm lediglich eine gültige SVN zugeordnet.

Objektorientierung als ideale Abbildung?

Mathematische Werte – Objekte als ideale Metapher?

- Ganze Zahlen → Zweierkomplement (`int`)
- Rationale Zahlen → Gleitkommazahlen (`float`)
- Boolesche Wahrheitswerte (`boolean`)

- Als Objekte implementieren?
  - Erzeugen einer „4“ erzeugt? Zustand einer „4“?
  - Wurde die Zahl Pi entdeckt oder erzeugt?
  - Wurde `true` vor `false` erzeugt? Ist die Reihenfolge wichtig?
  - Welches Verhalten haben solche Zahlen?

primitive Typen in Sprachen wie Java:
besser auf Werte als auf Objekte abbilden

Zahlen als Werte

- Zahlen als eine Form von Werten verwendet für:
  - zur Identifikation (Bezeichnung) von modellierten Gegenständen (Bsp.: Kontonummer, Personalnummer),
  - zum Abzählen und Ordnen von Gegenständen außerhalb und im Rechner,
  - zur Repräsentation von messbaren Größen.
- Im Bereich Naturwissenschaften und Technik:
  - zur numerischen Analyse, d.h. zur Lösung mathematischer Probleme durch Operationen auf Zahlen.

- Alle Zahlen sind Werte – aber sind alle fachlichen Werte auch Zahlen?

Fachliche Werte

- gehören oft zu Anwendungsbereich
- haben eine fachliche Bedeutung.
  - Beispiele: Postleitzahl, Bankleitzahl
- haben oft Operationen, die von den allgemeinen numerischen Operationen abweichen. Beispiele:
  - Kontonummern: konkatenieren (verketten) möglich (Bildung IBAN), subtrahieren nicht .
  - Geldbeträge: addieren u. subtrahieren, multiplizieren nicht .
- Nicht alle fachlichen Werte durch Zahlen sinnvoll abbildbar – Beispiele?

- (programmiersprachliche) Werte in Software sollten bestimmte Eigenschaften besitzen, die sie von Objekten unterscheiden.

Gegenüberstellung: Wert versus Objekt

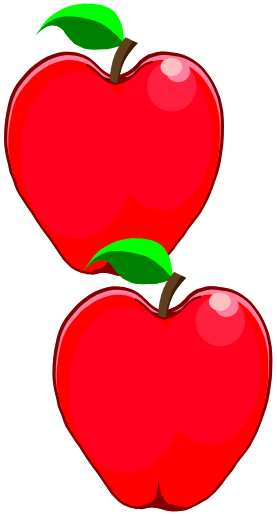
Zentrale Eigenschaften von Werten und Objekten? Unterschiede?

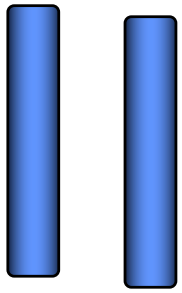
- Ein Wert ist:
  - abstrakt
  - zeitlos
  - unveränderlich

- Ein Objekt ist:
  - zustandsbehaftet (potenziell veränderbar)
  - erzeugbar und zerstörbar (existiert in Raum und Zeit)

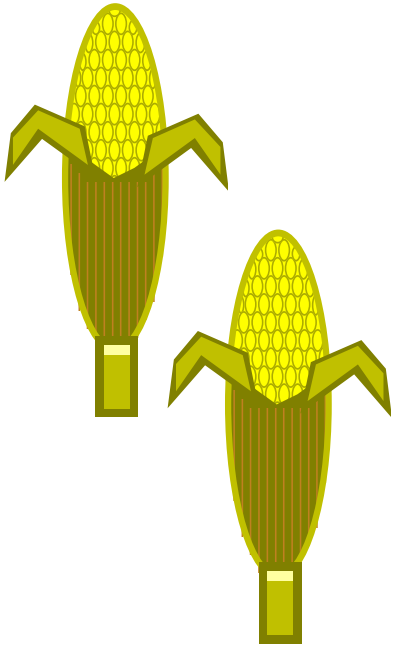
Wert: Abstrakt

- Ein Wert ist abstrakt:
  - Werte sind immer immateriell.
  - Werte abstrahieren von konkreten Kontexten.
  - Fachliche Werte sind häufig Abstraktionen von Dingen, um diese zu identifizieren (Kontonummer, Postleitzahl).





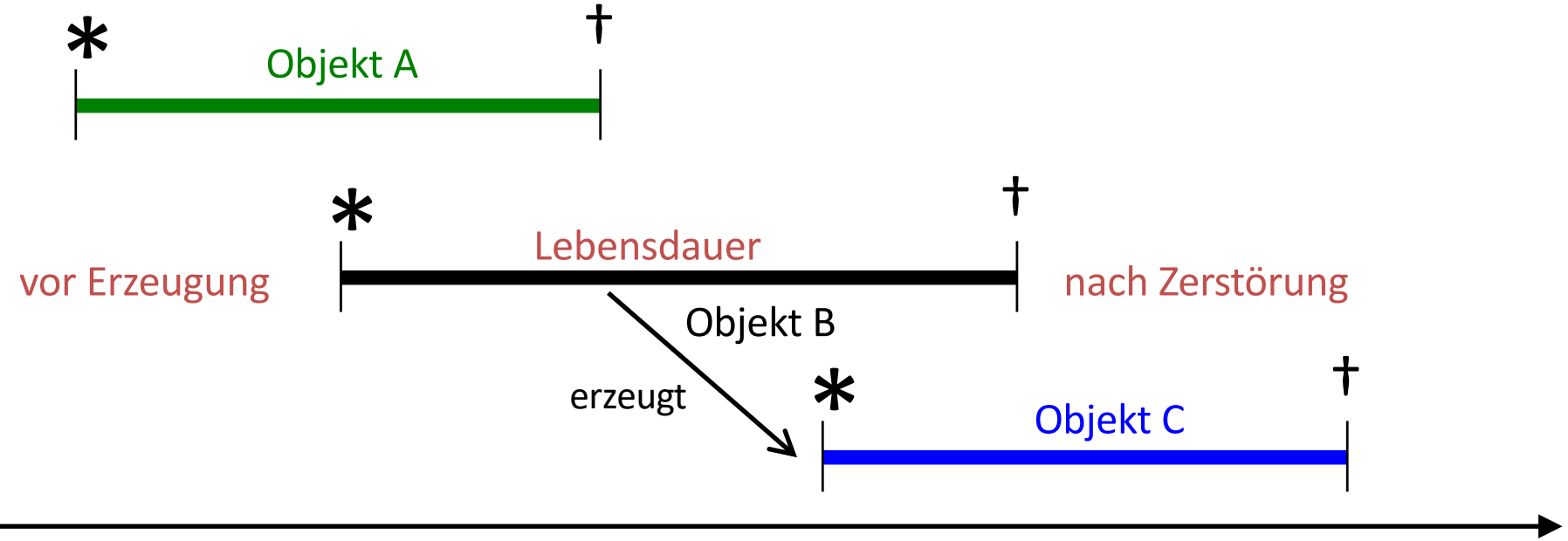
2



Objekt: Erzeugbar und zerstörbar

Ein Objekt kann erzeugt und zerstört werden:

- Objekte existieren in der Zeit, haben einen zeitlichen Anfang und ein Ende, also eine Lebensdauer.
- Verschiedene Zeitabschnitte: Vor Erzeugung eines Objektes , während seiner Lebenszeit und nach seiner Zerstörung.



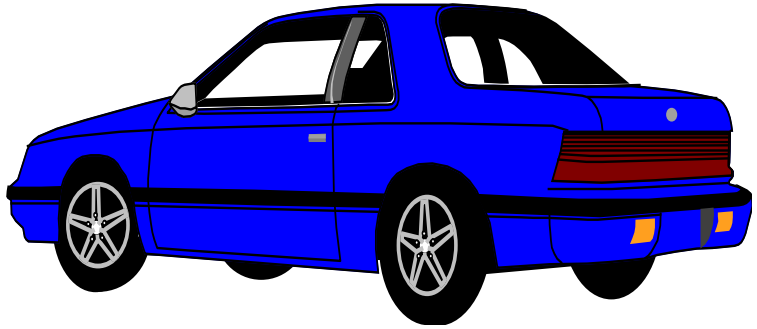
Wert: Zeitlos

- Ein Wert ist zeitlos:
  - Begriffe wie Zeit und Dauer sind nicht anwendbar.
  - Werte werden nicht erzeugt oder zerstört.
  - In Ausdrücken entstehen keine Werte und sie werden nicht verbraucht.

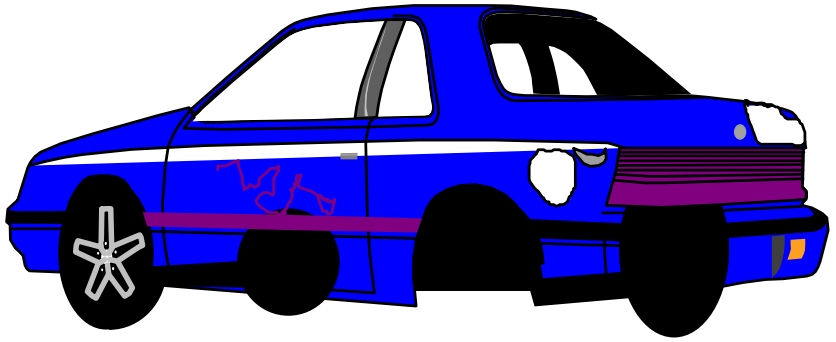
```
40 + 2 = 42
```

Objekt: Potenziell veränderbar

- Ein Objekt kann veränderbar sein:
  - die erkennbaren Merkmale können sich ändern,
  - trotz Veränderung bleibt die Identität erhalten.



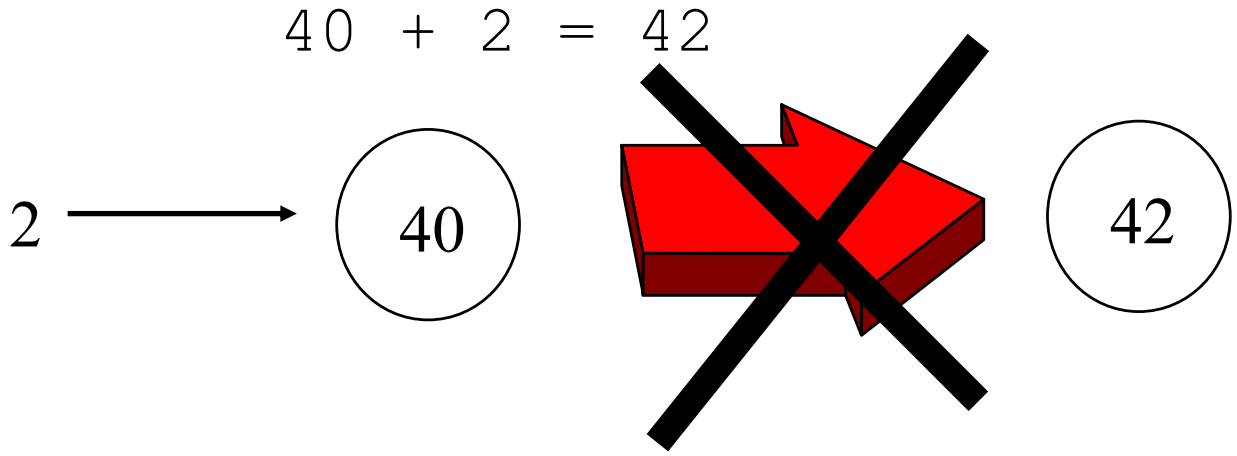
Mein Auto 2009



Mein Auto 2016

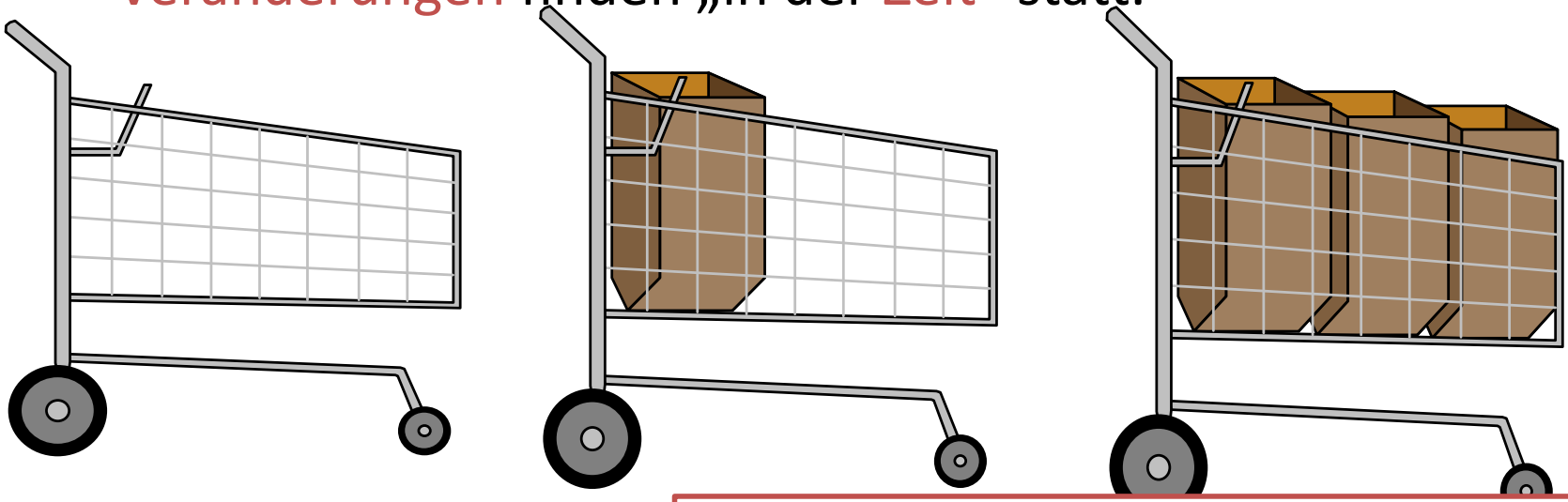
Wert: Unveränderlich

- Ein Wert ist unveränderlich:
  - Er kann berechnet und auf andere Werte bezogen werden, aber nicht verändert werden.
  - Funktionen können auf Werte angewandt werden, um andere Werte zu berechnen.



Objekt: Zustandsbehaftet

- Ein Objekt hat einen (inneren) Zustand:
  - Veränderung eines Objekts bedeutet die Veränderung seines Zustandes.
  - Veränderungen finden „in der Zeit“ statt.



„Mein“ Einkaufswagen

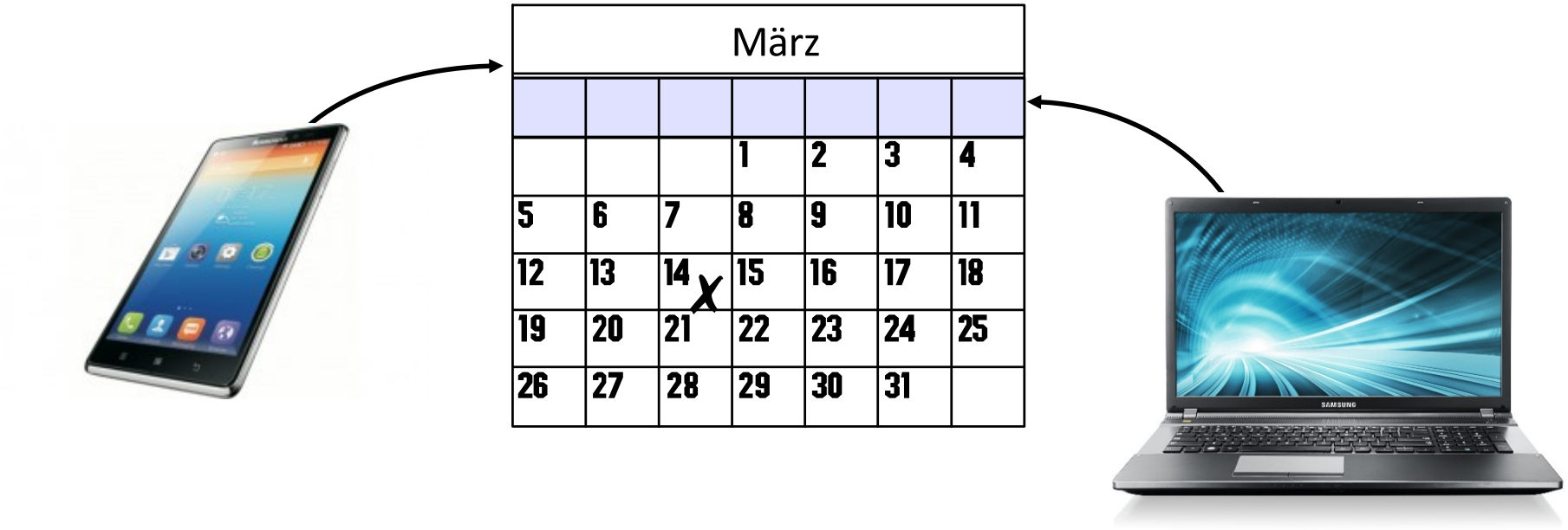
Es gibt auch unveränderliche Objekte: Diese erhalten ihren Zustand einmalig bei ihrer Erzeugung.

Wert: Zustandslos

- Ein Wert ist zustandslos:
  - Operationen berechnen Werte; sie verändern sie aber nicht.
  - Zeitbegriff auf Werte nicht anwendbar, können sich nicht „mit der Zeit“ verändern.
  - Zustände lassen sich zwar mit Hilfe von Werten modellieren (Kontostand), aber ein Wert selbst hat keinen Zustand.

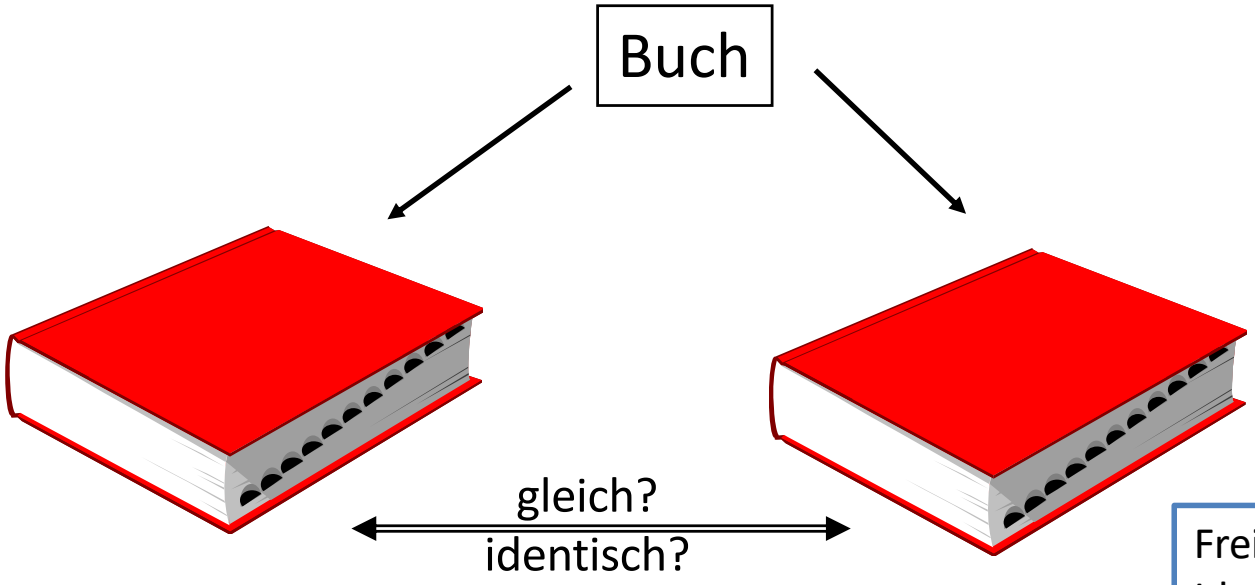
Objekt: Zur Kommunikation benutzbar

- Ein Objekt kann zur Kommunikation und Kooperation benutzt werden:
  - Verändert ein Benutzer den Zustand eines Objekts, ist dies für andere Benutzer erkennbar.
  - Dieser „Seiteneffekt" kann gewünscht oder problematisch sein.



Objekt: Unterschied zwischen Gleichheit und Identität

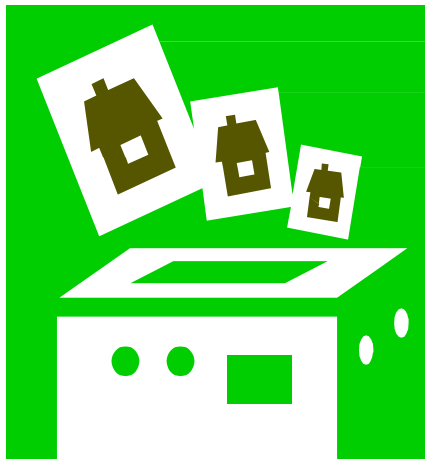
- Zwei gleiche Objekte sind nicht notwendig identisch:
  - Gleichheit bezieht sich auf die erkennbaren Merkmale,
  - Identität auf die (äußeren) Zusammenhänge eines Objekts (Position in Raum und Zeit).

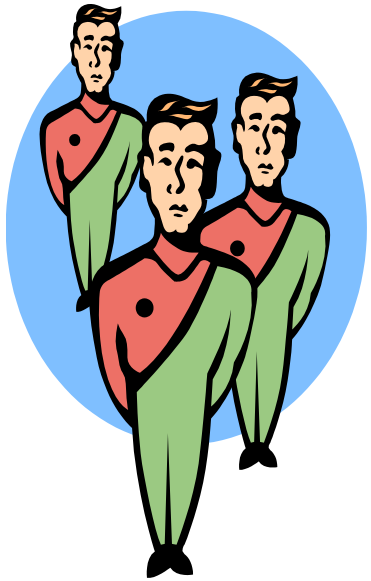


Frei nach Frege:
Identität ist die Beziehung, in der ein Objekt mit sich selbst und mit keinem anderen steht.

Objekt: Potenziell kopierbar

- Begriffe wie „Original“ und „Kopie" sind bei Objekten fachlich oft sinnvoll.





- Von Werten gibt es keine Kopien.
- Der Begriff Anzahl ist auf einen Wert nicht anwendbar.

Objekte und Werte unterscheiden – Objektorientierte Sprachen?

Unterscheidung zwischen Werten und Objekten wichtig - Grundkonzepte der Softwareentwicklung

- Objektorientierte Sprachen: mit Klassen beliebige **Objekttypen** definieren
  - Schlüsselwort `class` konstruiert neuen Typ: **Typkonstruktor**.
- Menge von **Werttypen** innerhalb einer Sprache üblicherweise festgelegt:
  - **Beispiel** Java: `int`, `byte`, `short`, `long`, `float`, `double`, `boolean`, `char`

Von Objekten zu Typen…

Objektorientierte Programmiersprachen:

- **Objekte** der realen Welt realisieren durch Definieren von **Klassen**, von denen Exemplare erzeugt werden können.
- Jede Klasse definiert einen **Typ**, die Operationen der Klasse sind auch die **Operationen** des Typs.
- Die Exemplare einer Klasse bilden (extensional) die **Elementmenge** ihres **Objekttyps**.

Wir nennen diese Menge explizit **Elementmenge,** da der sonst übliche Begriff „Wertemenge“ gerade bei der Diskussion über Werte und Objekte verwirrend sein kann. Sowohl Werte als auch Objekte bezeichnen wir im folgenden als **Elemente** ihres Typs.

Wikipedia: Die Extension eines Begriffs (z.B. „Mensch“) […] ist sein Umfang, das heißt die Menge aller Objekte („Erfüllungsgegenstände“), die unter diesen Begriff fallen.

…und von Werten zu Typen

- **Werte** werden in Programmiersprachen durch vordefinierte **Werttypen** (Java: ganzen Zahlen `int`, Wahrheitswerte `boolean`) modelliert.
- Die **Elementmenge** dieser Typen ist **unveränderlich**:
  - Werttyp `int`: 2^{32} Elemente, Werttyp `boolean:` zwei Elemente
- Allgemein: gilt: Wenn Werte konzeptuell nicht erzeugt und vernichtet werden, dann ist zwangsläufig die **Elementmenge bei Werttypen unveränderlich**.

Begriff „Konstruktor“ bei Werttypen deshalb vermeiden,

Besser: „**Selektor“** als Bezeichung von Operationen, die Werte eines Werttyps liefern. Sie selektieren einen Wert aus der Menge der bestehenden Werte.

Zwischenstand: Wert, Objekt und der Typbegriff

Ein **Wert** ist:
- abstrakt
- zeitlos
- unveränderlich

Ein **Objekt** ist:
- zustandsbehaftet (potenziell veränderbar)
- erzeugbar und zerstörbar (existent in Raum und Zeit)

in Programmiersprachen definiert über

Werttypen:
- haben eine unveränderliche Elementmenge
- bieten Selektoren an

Objekttypen:
- haben eine dynamische Elementmenge
- bieten Konstruktoren an

Gemeinsam:
- Typ = Elementmenge + Operationen
- Variablen des Typs mit veränderbarer Belegung

Die Frage der Gleichheit

Gleichheitsbegriff in Programmiersprachen:

- deutliche Unterschiede zwischen Werten und Objekten.
- Beispiel Java: Was bedeutet `a == b` ?
- Beispiel:

```
a = b;
if (a == b) { ... }
a = 42;
b = 42;
if (a == b) { ... }
```

Wir würden uns wünschen, dass `a` und `b` gleich sind!

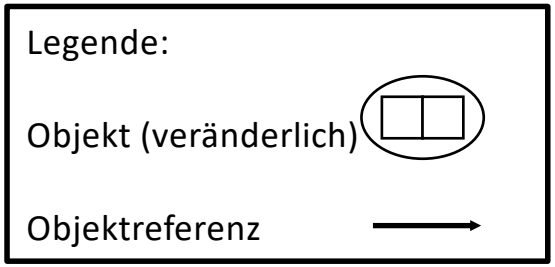
- Was ist dann mit Strings?
- Beispiel:

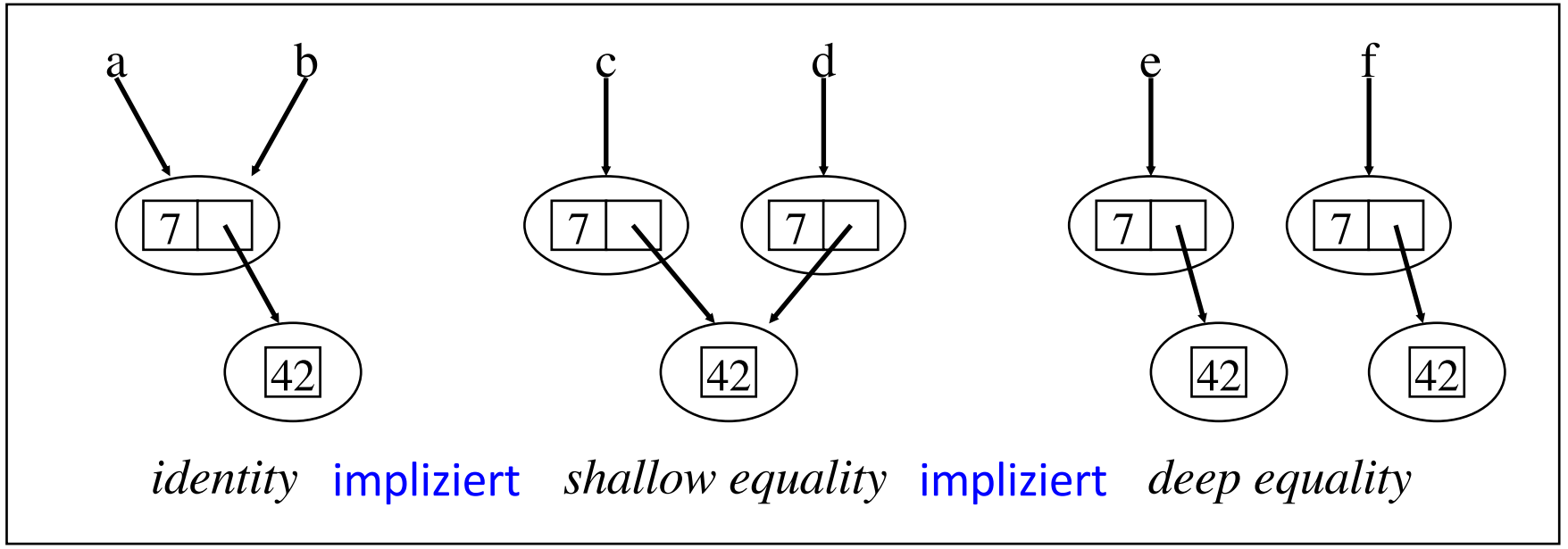
```
name = "Elling";
if (name.toLowerCase() == "elling") {
   ...
}
```

Gleichheit für Objekte

Technisch sind drei Formen unterscheidbar:

- Referenzgleichheit, Identität (identity)
- Einfache strukturelle Gleichheit (shallow equality)
- Rekursive strukturelle Gleichheit (deep equality)





Technische versus semantische Gleichheit

- Alle Formen technischer Gleichheit einfach zu implementieren, aber nicht für alle Fälle ausreichend.

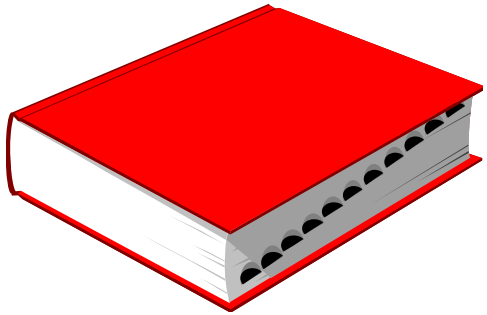
Beispiel:

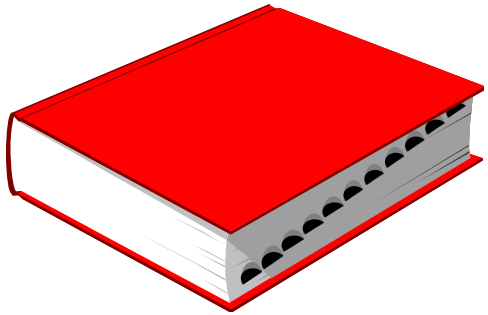
Bibliotheksbuch:

Titel Autor ISBN RefNummer

- kann als gleich angesehen werden, auch wenn die Referenznummer unterschiedlich ist...

➢ Konzept von **semantischer Gleichheit** außerdem benötigt





Selbst definierte Gleichheit mit `equals`

… mit `equals` in Java für jeden Objekttyp definierbar, wann Objekte als gleich gelten sollen

Ist das die Lösung?

Die Prüfung auf Gleichheit ist dann ein Aufruf einer Operation:

- asymmetrisch
- Vergleich mit `null`?
- Welchen Typ hat der Parameter?
  - `Object` wie in Java? Das führt zu Type Casts

Gleichheit in Theorie und Praxis

Mathematisch formuliert:

- Reflexiv
  a = a (Identität impliziert Gleichheit)
- Symmetrisch
  a = b => b = a
- Transitiv
  a = b & b = c => a = c

Merken:
Bei Objekten besteht ein Unterschied zwischen Gleichheit und Identität; bei Werten hingegen ist diese Unterscheidung unüblich.

Transitivität seit Java 1.5...

Auto-Boxing und -Unboxing:

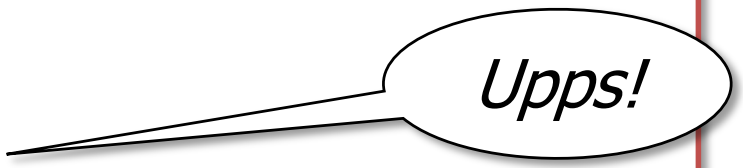
```
Integer a = 128;
int b = 128;
Integer c = 128;
```

Es gilt:

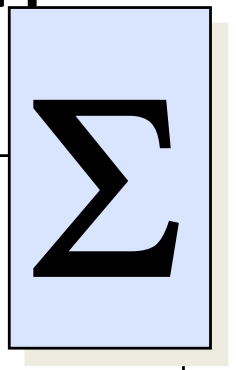
```
a == b
b == c
```

Aber:

```
a != c
```



Fazit: Wert, Objekt und der Typbegriff



Ein **Wert** ist:
- abstrakt
- zeitlos
- unveränderlich

Ein **Objekt** ist:
- zustandsbehaftet (potenziell veränderbar)
- erzeugbar und zerstörbar (existent in Raum und Zeit)

in Programmiersprachen definiert über

Werttypen:
- haben eine unveränderliche Elementmenge
- bieten Selektoren an

Objekttypen:
- haben eine dynamische Elementmenge
- bieten Konstruktoren an
- Gleichheit und Identität unterscheiden

Gemeinsam:
- Typ = Elementmenge + Operationen
- Variablen des Typs mit veränderbarer Belegung

Aufgepasst: Unveränderliche Objekte versus Werte

- In vielen Anwendungskontexten Definition von **unveränderlichen Objekten** sinnvoll

Beispiel:

- Für Film in Filmdatenbank sind **Titel**, **Regisseur** und **Filmlänge** unveränderlich

➢ ein Film kann durch eine Objektklasse **Film** definiert werden, deren Exemplare diese Eigenschaften bei der Erzeugung eines Filmobjektes übergeben bekommen und sie anschließend nur über lesende Operationen zur Verfügung stellen.

Wenn ein **Objekt ausschließlich sondierende Operationen** anbietet – ist es dann nicht quasi ein Wert?

- **Nein**: Für einen Wert müssen **alle konzeptionellen Eigenschaften** zutreffen.
- Beispiel Film Abstrakt, kann auch als Unveränderlich angesehen werden; aber nicht zeitlos, denn es gibt einen Entstehungszeitpunkt

Zukünftige Sprachen: benutzerdefinierte Werttypen

Zukünftige objektorientierte Programmiersprachen sollten ein dediziertes **Konstrukt** anbieten, mit dem eigene **Werttypen** definiert werden können:

- Ihre **Wertemenge** sollte **fixiert** sein; da Werte nicht erzeugt und zerstört werden können, kann es beispielsweise keine Konstruktoren für Werte geben.
- Ihre **Elemente** müssen **unveränderlich** sein (am besten garantiert durch den Compiler).
- Ihre **Operationen** sollten **referentiell transparent** sein und **keine beobachtbaren Seiteneffekte** zeigen.
- Sie sollten **ausschließlich auf der Basis anderer Werttypen definiert** werden (Werte können sich nicht auf Objekte beziehen).

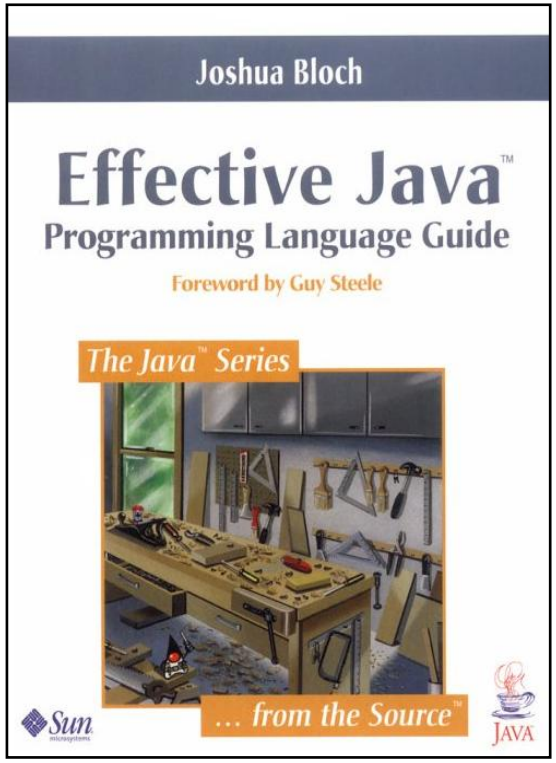
In manchen Sprachen: `structs`

- Wie Objekte
  - Können Zustandsvariablen und Methoden definieren wie sonst Klassen
  - Structs werden „erzeugt“
  - Elemente der Wertemenge sind änderbar
- Wie Werte
  - Struct-Variablen halten nicht Referenz, sondern die Werte des Struct-Examplars
  - Bei Zuweisungen/Übergabe als Parameter wird Inhalt der Variable kopiert
  - Konstante struct-Variablen können ihren Zustand nicht ändern

Werttypen in Java mit Wertklassen

Java ohne explizite Unterstützung für benutzerdefinierte Werttypen

- Werttypen **mit Objektklassen definieren** (im Folgenden **Wertklassen** genannt) und dabei darauf achten, dass die Werteigenschaften eingehalten werden.
- mehrere **Programmiermuster** unterstützen dabei
- Prominentes Beispiel:
  - Das **Typesafe Enum Pattern** von Bloch.



Das Typesafe Enum Pattern

- Beschreibt, wie **Aufzählungstypen** typsicher in Java modelliert werden können.

Grundmuster:

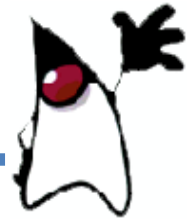
- Biete ein `public static final` Feld für jede Konstante der Aufzählung.
- Verstecke den Konstruktor.

```
public class Farbe {
    private final String _name;

    private Farbe(String name) { _name = name; }

    public String toString() { return _name; }

    public static final Farbe ROT = new Farbe("rot");
    public static final Farbe GRUEN = new Farbe("gruen");
    public static final Farbe BLAU = new Farbe("blau");
    ...
}
```

Ist seit Java 1.5 eingebaut: mit **Enums** in Java wird dieses Muster realisiert.

- allerdings keine „waschechten“ Werttypen, da an den Exemplaren eines Enums ein **veränderlicher Zustand** modelliert werden kann.
- Es entstehen **Wertobjekte** – fachlich motivierte Werte, die technisch durch Objekte repräsentiert werden müssen.
- Frage: Für welche Wertemengen ist das sinnvoll?

Sechs Richtlinien zu Wertklassen in Java



Konstruktion von Wertklassen

Stelle sicher,

1. dass Wertobjekte keinen veränderbaren Zustand haben;
2. dass Werte und Wertobjekte sich nicht auf Objekte beziehen;
3. dass im Quelltext einer Wertklasse keine (bestehenden) Objekte verändert werden.

4. Verberge die (technisch notwendige) Erzeugung von Wertobjekten.
5. Implementiere `equals` und dazu passend `hashCode`.

Benutzung von Wertklassen

6. Verwende immer `equals` statt `==` bei der Prüfung auf Gleichheit zweier Werte.

1. Unveränderlicher Zustand

- Die der Zustand von Exemplaren der Wertklassen sollte unveränderlich sein.

- **Sprachunterstützung**
  - Mit dem Schlüsselwort `final` kann für **Zustandsfelder** festgelegt werden, dass sie unveränderlich sein sollen.
    - Einem Zustandsfeld, das als `final` gekennzeichnet ist, darf nur innerhalb des Konstruktors, oder direkt bei der Deklaration ein Wert zugewiesen werden.
  - Mit dem Schlüsselwort `final` kann für die gesamte **Wertklasse** festgelegt werden, dass es keine Subklassen geben darf.
    - Auf diese Weise wird verhindert, dass Unterklassen mit veränderlichem Zustand definiert werden können, deren Exemplare polymorph verwendet werden könnten.

2. Wertobjekte als Blätter im Objektbaum

- Die primitiven Werte in Java bilden bereits die Blätter im Objektbaum. Darüber hinaus sollten Wertobjekte keine Referenzen auf (echte) Objekte enthalten.
- **Sprachunterstützung**
  - keine
- **Selbst beachten**
  - In einer Wertklasse sollten die Typen aller Zustandsfelder nur Werttypen (primitive Typen oder Wertklassen) sein.
- String kann als eine Wertklasse angesehen werden.

3. Keine bestehenden Objekte verändern

- Wertobjekte sollten keine (echten) Objekte verändern.
- **Sprachunterstützung**
  - keine
- **Selbst beachten**
  - Im Quelltext einer Wertklasse sollten keine (bereits bestehenden) Objekte verändert werden.
  - In der Schnittstelle einer Wertklasse sollten deshalb keine Objekttypen als Parametertypen erscheinen.
  - Pragmatisch kann zugelassen werden, dass in einer Methode lokal Objekte erzeugt werden (`StringBuffer`, `Formatter`, etc.).

4. Erzeugung verbergen

- Die Kontrolle über die Erzeugung von Exemplaren einer Wertklasse sollte in der Wertklasse selbst liegen.
- **Sprachunterstützung**
  - Die Konstruktoren einer Klasse können als `private` deklariert werden. Klienten können dann nicht mehr direkt Exemplare der Klasse erzeugen.
  - Als Ersatz kann eine Klasse **Fabrikmethoden** anbieten: Klassenmethoden (mit `static` deklariert), die Exemplare der Wertklasse liefern. Diese Methoden dienen dann als **Selektoren**.
- Auf diese Weise ist es teilweise möglich, auf Regel 5 zu verzichten:
  - Wenn in der Wertklasse garantiert wird, dass für jeden Wert nur ein Wertobjekt erzeugt wird, dann können Klienten Referenzgleichheit verwenden (siehe die Enumerations in Java).

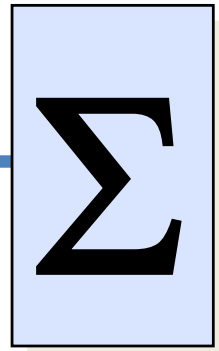
5. equals und hashCode implementieren

- Die Gleichheit für Wertobjekte sollte explizit definiert werden. Da verschiedene Wertobjekte denselben Wert repräsentieren können (keine Referenzgleichheit), muss die Gleichheit mit der Operation `equals` definiert werden.
- **Sprachunterstützung**
  - keine
- **Selbst beachten**
  - Der Vertrag des Typs `Object` für seine Operationen `equals` und `hashCode` muss eingehalten. U.a. gilt:
    - Zwei Wertobjekte, die als gleich gelten, müssen auch denselben Wert als Hash-Code liefern.

SE-Bar um Produkt-ids erweitern

- SE-Bar wird aufgekauft
- Produkte müssen in das zentrale System aufgenommen werden
- Ids immer fünfstellig
- Bereich zwischen 10000 - 50000 für Getränke reserviert
- ab 50001 für Essen
- ab 80000 sonstige Produkte

Zusammenfassung Wert und Objekt



- Werte und Objekte sind fundamental verschiedene Konzepte.
- Zustandsbehaftete, vergängliche Gegenstände und Konzepte lassen sich gut als Objekte modellieren.
- Zeitlose und unveränderliche Größen wie Zahlen und Zeiträume werden sinnvoll als Werte dargestellt.
- Fachliche Werte spielen in vielen Anwendungsbereichen eine große Rolle. Sie haben oft Operationen, die nicht den mathematischen Operationen auf Zahlen entsprechen.
- Objektorientierte Programmiersprachen stellen neben Objekten auch elementare Werte zur Verfügung. Sie bieten aber keine einfachen Möglichkeiten, benutzerdefinierte fachliche Werte einzuführen.

Literaturhinweise

B.J. MacLennan, *Values and Objects in Programming Languages*, ACM SIGPLAN Notices, Vol. 17, No. 12, Dec. 1982.

[Grundlage für diesen Teil]

J. Bloch, *Effective Java Programming Language Guide, 2nd Ed.*, Addison Wesley, 2008.

[Kenntnisreiche Darstellung der Fußangeln von Java; ein Muss für professionelle Java-Entwickler]

Online verfügbar: http://www.pascal-man.com/navigation/faq-java-browser/java-concurrent/Effective%20Java%20-%20Programming%20Language%20Guide.pdf