Objektorientierte Programmierung
Grundbegriffe

Objektorientierung (1)

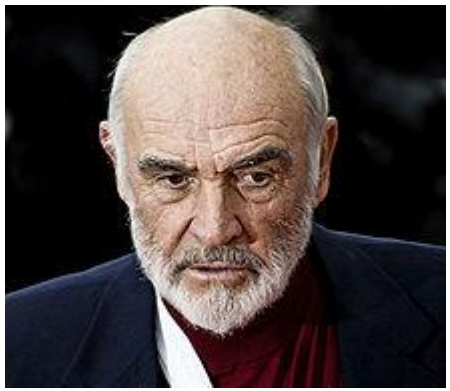
- Im Bereich der Programmierung wird von Objektorientierung gesprochen, wenn zumindest drei Konzepte unterstützt werden:
  - Datenkapselung (encapsulation, data/information hiding)
  - Polymorphismus (polymorphism)
  - Vererbung (inheritance)
- Weitere Konzepte sind oft vorhanden, müssen aber nicht unbedingt zur Verfügung gestellt werden:
  - Assertions
  - Exceptions
  - Closures
  - …

Objektorientierung (2)

- Objektorientierte Programmierung wird häufig als "Programmierung mit Klassen" umschrieben.
- Für viele objektorientierte Programmiersprachen ist das korrekt.
- Es ist aber auch möglich, die objektorientierten Prinzipien ohne Klassen umzusetzen. Zum Beispiel in der Prototyp-basierten Programmierung (prototype based programming), die etwa in JavaScript verwendet wird.

Objekt (object)

- Objekte sind Datenstrukturen, die (physikalische oder konzeptuelle) Objekte aus dem zu implementierenden System widerspiegeln. Objekte haben einen **Zustand** (state) - beschrieben durch Instanzvariable (instance variable) - und ein **Verhalten** (behavior) - beschrieben durch Methoden (methods).
- Objekte werden in Klassen zusammengefasst.



Zustand:

Name: Sean Connery
Geburtsdatum: 25.8.1930
…

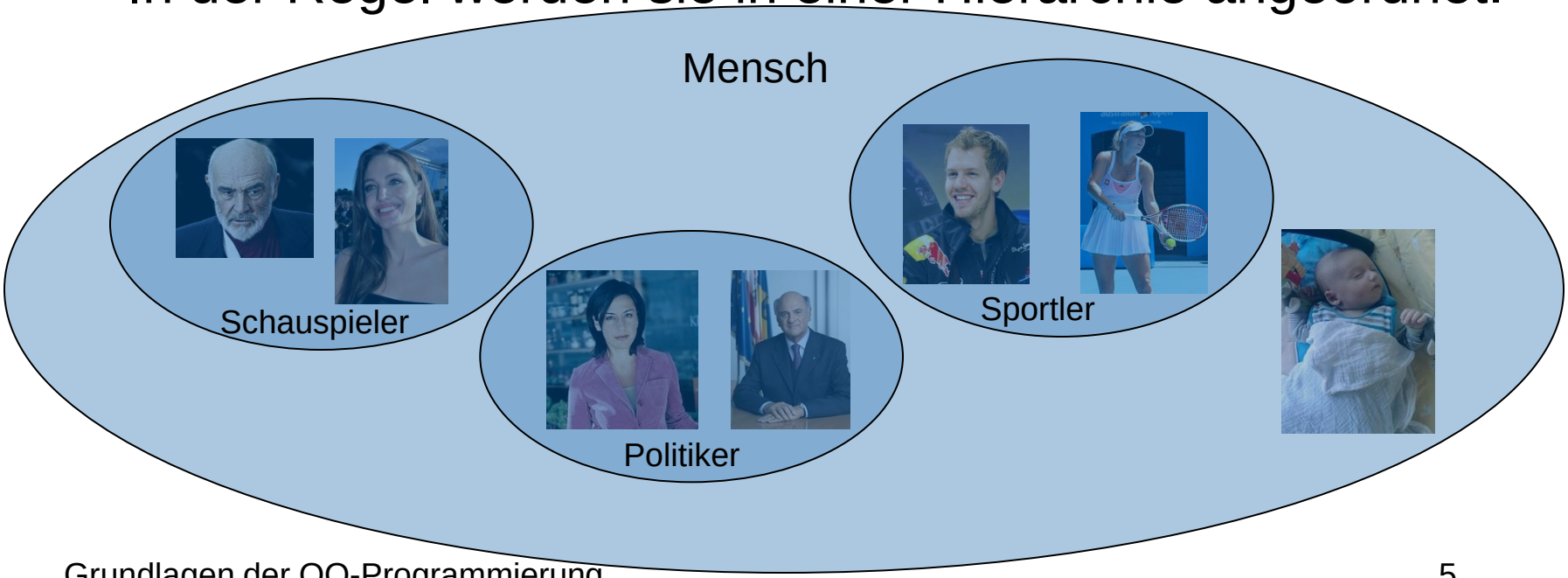
Verhalten:

isst (was, wieviel, …**)**
läuft(wie schnell, richtung …**)**
schläft()
…

Verhalten ändert in der Regel den Zustand

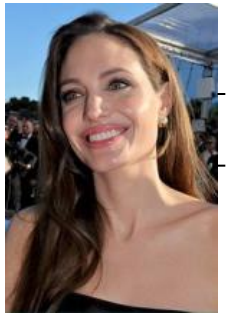
Klasse (class)

- Klassen umfassen die allen enthaltenen Objekten gemeinen Eigenschaften und können (abhängig von der verwendeten Programmiersprache) selbst wieder als Objekte betrachtet werden.
- In der Regel werden sie in einer Hierarchie angeordnet.



Instanz (instance)

- Von der Sicht der Klassen aus betrachtet, wird ein Objekt als eine Instanz (selten auch als Objektinstanz) einer Klasse bezeichnet.
- Daher rührt die Bezeichnung Instanzvariablen (jede Instanz hat ihren eigenen Zustand und somit eine eigene Menge von Zustandsvariablen).
- Das Erzeugen neuer Objekte wird auch als Instanziierung (instantiation) bezeichnet.



«instanceOf»-> Mensch

«instanceOf»-> Schauspielerin

Achtung:
Schauspielerin ist
keine Instanz von Mensch

Beziehungen zwischen Klassen

- *Mensch* ist die **Superklasse** (super class; auch Oberklasse oder Basisklasse) von *Schauspielerin*.
- *Schauspielerin* ist eine **Subklasse** (subclass; auch Unterklasse) von *Mensch*.
- Subklassen- und Superklassenbeziehungen können auch über mehrere Ebenen bestehen (z.B. *Mensch – VIP – Schauspielerin*)
- Bezüglich der Instanzen herrscht eine Teilmengenbeziehung (d. h., die Menge der Instanzen der Subklasse ist eine – nicht notwendigerweise echte - Teilmenge der Menge der Instanzen der Superklasse).
- In der Regel ist ein Objekt eine Instanz genau einer Klasse, die keine weiteren Subklassen mehr besitzt.

Abstrakte Klasse (abstract class)

- Eine Klasse, für die Objekte nicht direkt instanziiert werden können, heißt **abstrakte Klasse**.
- Die abstrakte Klasse sammelt die Gemeinsamkeiten aller Unterklassen (Eigenschaften und Verhalten), die dann nur einmal behandelt werden müssen.
- So wäre z.B. *Fahrzeug* eine typische abstrakte Klasse. Sie könnte *aktuelle* und *maximale Geschwindigkeit* als Attribute und *starten()*, *beschleunigen()* und *bremsen()* als Operationen enthalten.
- Es gibt aber kein Objekt das nur ein *Fahrzeug* ist. Sondern nur Objekte der (konkreten) Unterklassen *Auto, Bahn, Fahrrad*, …

Probleme mit Klassenhierarchie

- Was passiert, wenn ein Objekt mehreren (Sub)Klassen angehört, die nicht in einer Superklassen-Subklassen-Beziehung stehen?
- Mögliche Lösungen, je nach Programmiersprache und exakter Aufgabenstellung
  - zu unterschiedlichen Zeitpunkten (z.B. Schauspieler wird Politiker)
    - Objektmigration (object migration)
    - Rollen (role) z.B. Perl 6
  - zum selben Zeitpunkt (z.B. schauspielernde Sportlerin)
    - Mehrfachvererbung (multiple inheritance) z.B. C++
    - Schnittstellen (interface) z.B. Java
    - Komplexe Klassenhierarchien z.B. Smalltalk
    - Delegation (delegation) z.B. JavaScript

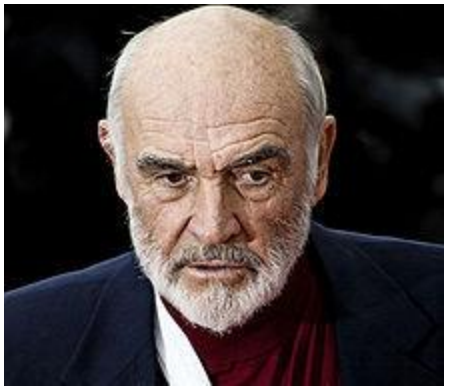
Abstraktion (abstraction)

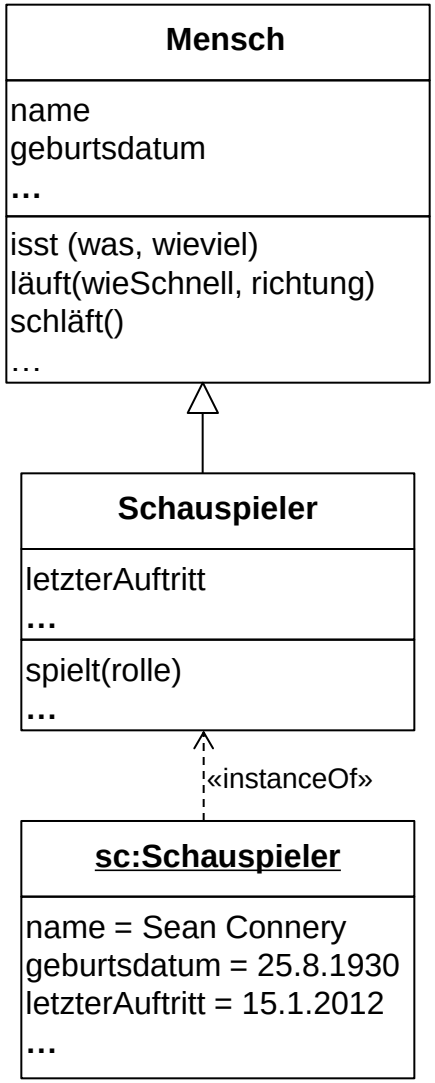
- Bei der objektorientierten Programmierung werden die Objekte der realen Welt zumeist durch entsprechende Datenobjekte repräsentiert.
- Da nicht alle Eigenschaften des realen Objekts exakt erfasst werden können, beschränkt man sich darauf, nur die wesentlichen Eigenschaften in den Datenobjekten aufzunehmen. Dies wird als **Abstraktion** bezeichnet.
- Der Schritt in der umgekehrten Richtung heißt **Konkretisierung** (concretization).

Modellierung (modeling)

- Erstellen einer (in der Regel abstrakteren) Repräsentation der interessierenden (oftmals physikalischen) Realität. Das Modell erlaubt es, sich auf die für die jeweilige Aufgabenstellung wirklich wichtigen Details zu konzentrieren.
- Es ist nicht ungewöhnlich mehrere, unterschiedliche (manchmal sogar widersprüchliche) Modelle ein und desselben Sachverhalts zu verwenden.
- In der objektorientierten Programmierung gibt es zumindest ein konzeptuelles Modell (UML) und eine Implementierung (Programm).
- Beide Modelle beschreiben nicht nur die Objekte, sondern auch deren Wechselwirkungen.
- Das Programm kann als ein Modell betrachtet werden, das es erlaubt, bestimmte Bereiche der repräsentierten Realität zu simulieren.

Abstraktion

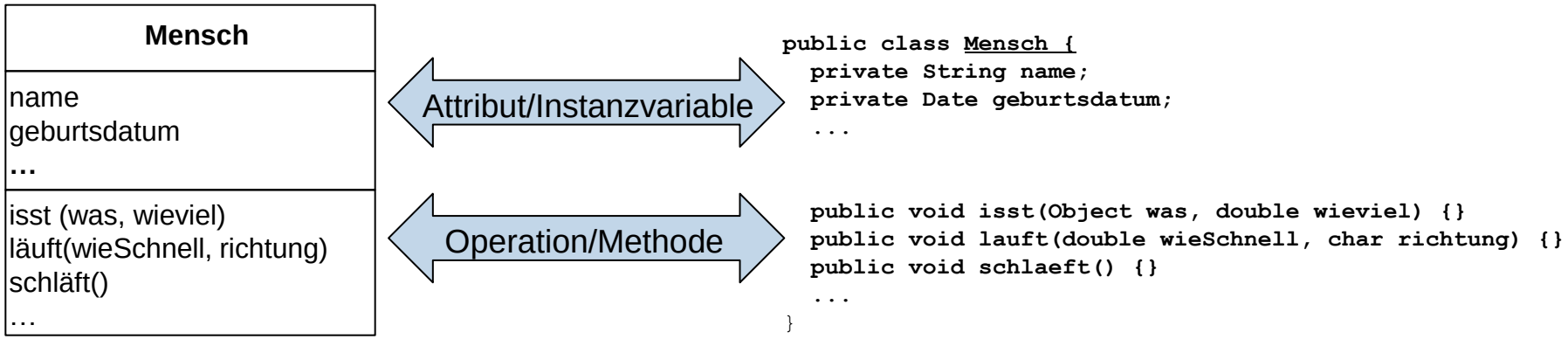




Konkretisierung

```
public class Mensch {
  private String name;
  private Date geburtsdatum;
  ...

  public void isst(Object was, double wieviel) {}
  public void lauft(double wieSchnell, char richtung) {}
  public void schlaeft() {}
  ...
}

public class Schauspieler extends Mensch {
  private String letzterAuftritt;
  ...

  public void spielt(String rolle) {}
  ...

  public static void main(String[] args) {
    Schauspieler sc=new Schauspieler("Sean Connery",
                                     "25.8.1930","15.1.2012");
  }
}
```

Objektzustand und -verhalten



- Die Begriffe **Methode** (method) und **Operation** (operation) bzw. **Instanzvariable** (instance variable) und **Attribut** (attribute) werden oft synonym verwendet. Die ersteren Begriffe werden bei der Implementierung bevorzugt, die zweiteren beim konzeptionellen Modell.
- **Eigenschaft** (property) wird ebenfalls oft als Synonym für Attribut verwendet, hat aber in manchen Programmiersprachen (z.B. C#) spezielle Bedeutung.

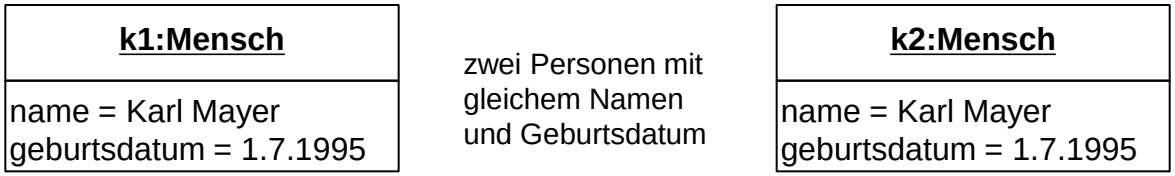
Lebenszyklus (life cycle) von Objekten

- Instanziierung - "Geburt"
- Existenz des Objekts – "Leben"
- Zerstörung (destruction) – "Tod"

- So lange das Objekt existiert, kann es seinen Zustand und eventuell auch die Klassenzugehörigkeit ändern, es ist aber immer dasselbe Objekt (die **Identität** ändert sich nicht).
- Wann und wie Objekte zerstört werden, hängt von der jeweiligen Programmiersprache ab. In Java: automatisch, wenn das Objekt nicht mehr benötigt wird (**garbage collection**).

Objektidentität (object identity)

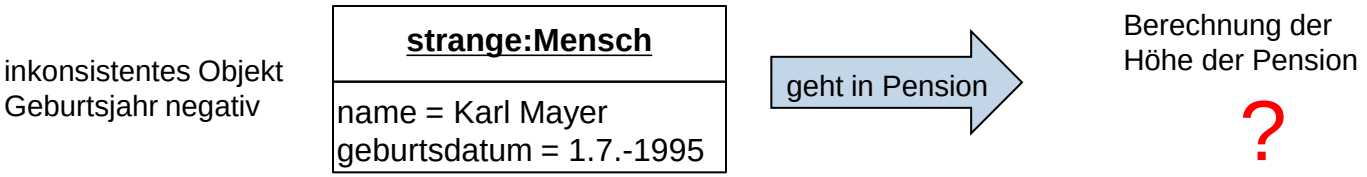
- Es muss sehr genau unterschieden werden zwischen **gleichen** Objekten (Übereinstimmung in allen, relevanten Instanzvariablen) und **identen** Objekten (ein und dasselbe Objekt tritt mehrfach auf).



- Kauft k1 eine Jahreskarte für öffentliche Verkehrsmittel, so ist es wichtig, ihn von k2 zu unterscheiden.
- Objektorientierte Sprachen stellen daher unterschiedliche Vergleiche zwischen Objekten zur Verfügung (z.B. in Java: == für Identität und *equals()* für Gleichheit).

Konsistenz (consistency) von Objekten

- Ein Objekt ist konsistent (gültig), wenn alle Instanzvariablen nur erlaubte Werte beinhalten.
- Ein konsistentes Objekt befindet sich in einem real möglichen Zustand.
- Inkonsistente Objekte können in der weiteren Verarbeitung zu schweren Problemen führen.



- Bedingungen, die erfüllt sein müssen, damit ein Objekt konsistent ist, können sehr komplex sein. Sie werden als Konsistenz- bzw. Integritätsbedingungen bezeichnet (consistency/integrity constraint/condition).

Datenkapselung (1)

- Das Verbieten der direkten Manipulation der Attribute eines Objektes durch beliebige andere Objekte wird als Datenkapselung bezeichnet.
- Die Zugriffsrechte können für jedes Attribut durch Definition der sogenannten Sichtbarkeit (visibility) festgelegt werden.
  - \- privat (Zugriff nur von derselben Klasse aus; private)
  - \+ öffentlich (Zugriff für alle erlaubt, public)
  - # geschützt (Zugriff nur von der Klasse und ihren Subklassen; protected)
  - ~ paket (Zugriff von allen Klassen im selben Paket; packet)

In Java können nicht nur erbende Klassen zugreifen, sondern auch alle im selben Package

Kein Schlüsselwort für Paketzugriff

```
public class Klasse {
  public String attribut1;
  private Date attribut2;
  protected char attribut3;
  double attribut4;
}
```

Klasse
+attribut1 - attribut2 #attribut3 ~attribut4 ...

Datenkapselung (2)

- Durch Verwendung von Operationen können beliebige, andere Objekte noch immer Änderungen von nicht öffentlichen Attributen bewirken. Die Operationen können aber dafür sorgen, dass keine inkonsistenten Zustände auftreten.
- Für Operationen können analog Sichtbarkeiten definiert werden.
- Faustregel: Attribute privat und Operationen öffentlich

- Anmerkung: Aus Effizienzgründen ist die Sichtbarkeit in den meisten objektorientierten Systemen auf Klassen und nicht auf einzelne Objekte bezogen. Ein Objekt der Klasse *Mensch* kann somit z.B. auf das private Geburtsdatum eines anderen Objekts der Klasse *Mensch* direkt zugreifen.

Datenkapselung (3)

- Neben der Möglichkeit, die Integrität von Objekten zu gewährleisten hat die Datenkapselung weitere Vorteile:
  - Einfacheres Debuggen: Der Code in dem ein bestimmter Fehler zu suchen ist, kann sich nur in den Methoden der jeweiligen Klasse befinden
  - Optimierungen: Instanzvariable, deren Wert aufwendig zu bestimmen ist, müssen erst berechnet werden, wenn darauf mittels einer Methode zugegriffen wird.
  - Effiziente Implementierung von abgeleiteten Attributen (derived attribute). Das sind solche, die aus anderen Attributwerten bei Bedarf berechnet werden können. Z.B. kann das *Alter* aus dem *Geburtsdatum* bestimmt werden. Die Instanzvariable für *Alter* kann eingespart werden.

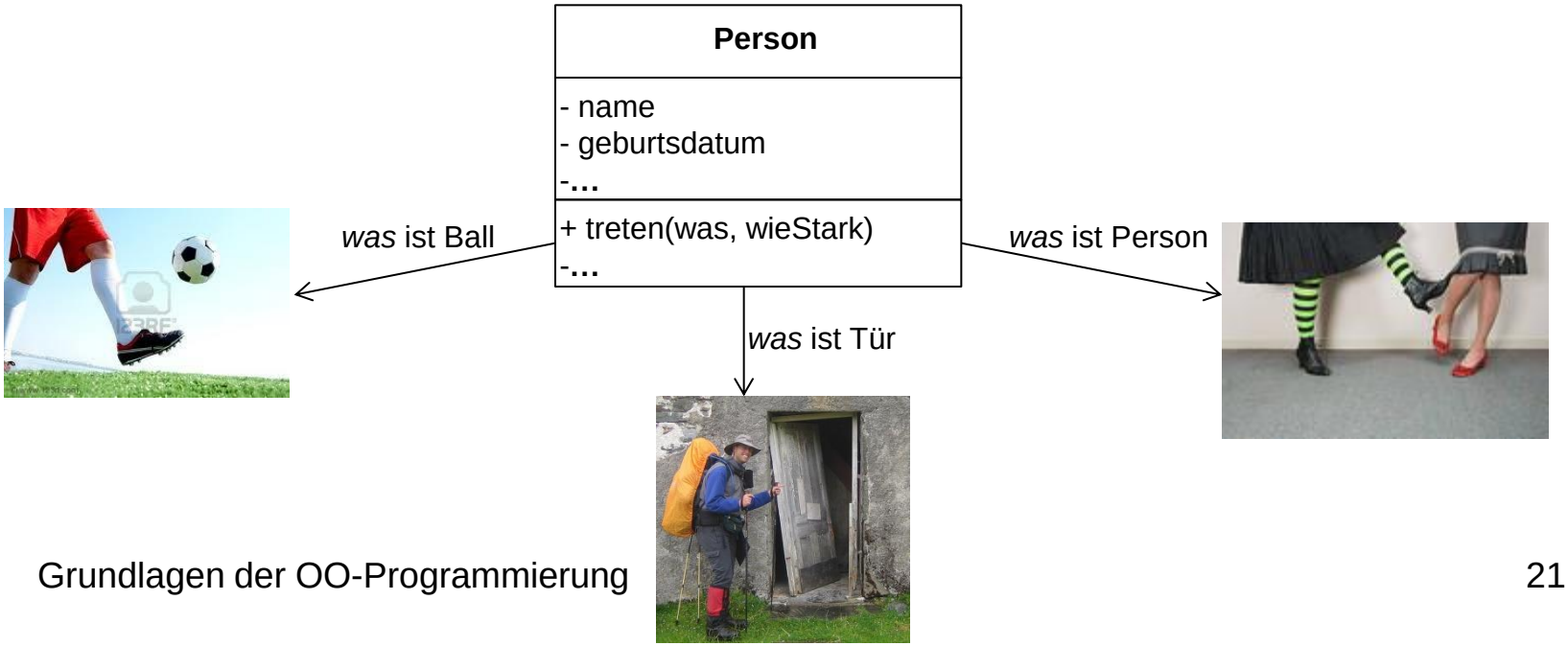
Datenkapselung (4)

- Bei sorgfältiger Verwendung der Datenkapselung darf jede Methode davon ausgehen, dass alle Objekte in einem konsistenten Zustand sind.
- Andererseits hat jede Methode die Verpflichtung, alle Objekte, die sie direkt verändern kann, am Ende wieder in einem konsistenten Zustand zu hinterlassen.

Polymorphismus

- Polymorphismus: grch. Vielgestaltigkeit

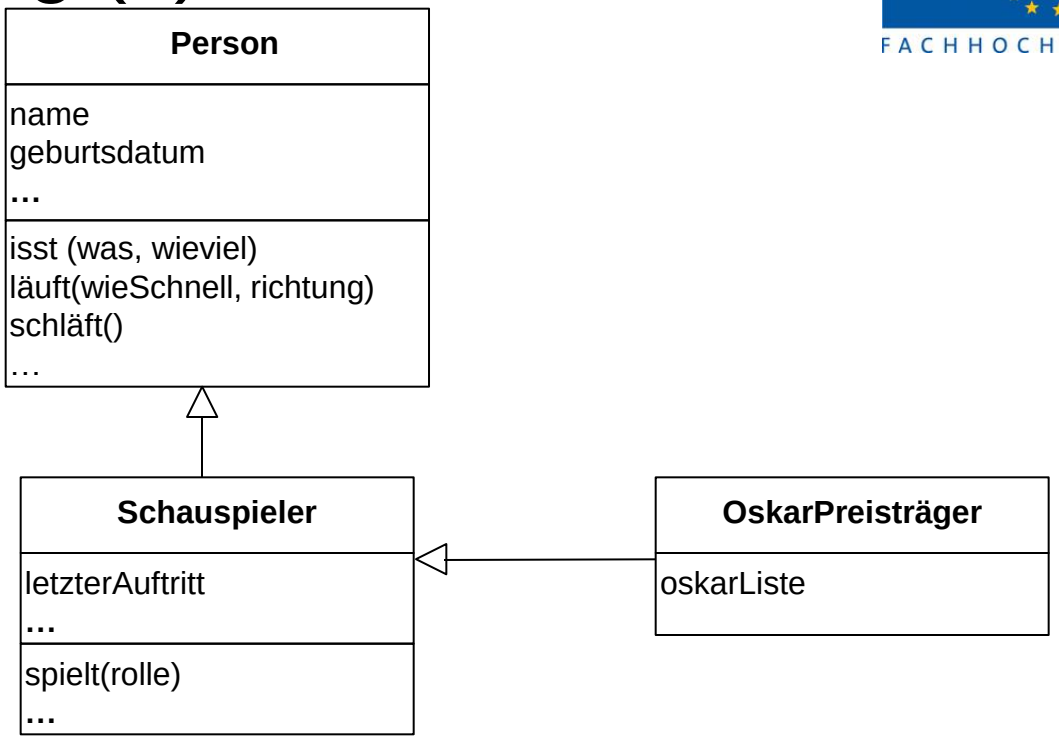
- Die gleiche Operation kann völlig unterschiedliche Auswirkungen haben, je nachdem, für welches Objekt sie durchgeführt wird.



Vererbung (1)

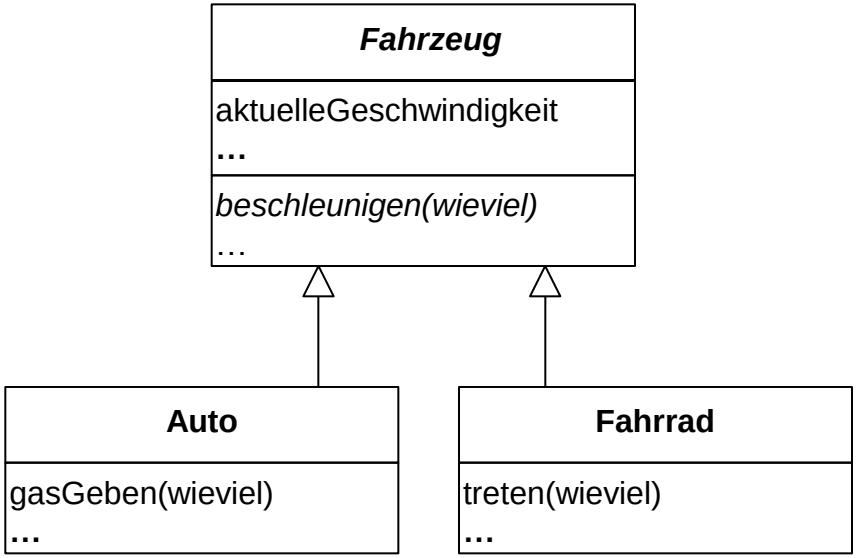
- Durch Vererbung werden alle Attribute und Operationen von der Superklasse auf die Subklassen übertragen.
- In der jeweiligen Subklasse müssen nur mehr die zusätzlichen Attribute und Operationen definiert werden.
- Eventuell können Attribute und Operationen für die jeweilige Subklasse adaptiert werden.

Vererbung (2)



- Liskovs Substitutionsprinzip (Liskov's substitution principle): Ein Objekt einer Subklasse kann in jedem Kontext verwendet werden, in welchem ein Objekt einer seiner Superklassen benötigt wird.
- Umkehrung gilt **nicht**!

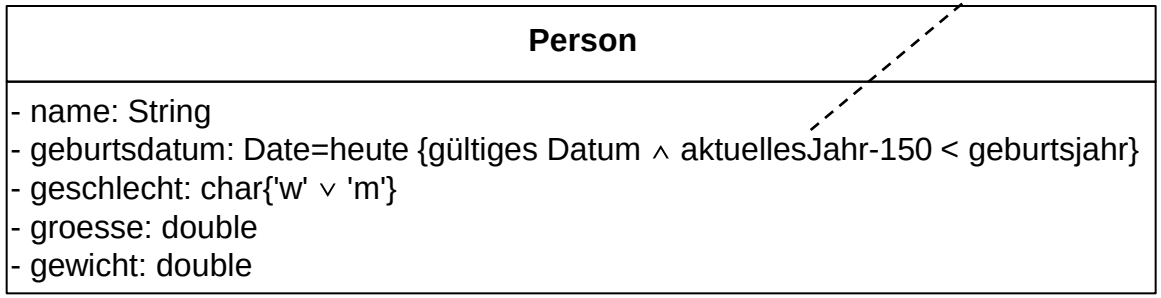
Vererbung und Polymorphismus



- Wird für ein Fahrzeug (dessen Typ beim Schreiben des Programms unbekannt ist) *beschleunigen()* aufgerufen, so führt das zum Aufruf der korrekten Methode mit entsprechend angepassten Parameterwerten. Also *gasGeben()*, falls es sich um ein Auto handelt, *treten()* für ein Fahrrad.

Klassen und Objekte in Java

Klasse Person



Integritätsbedingungen können in Java nicht direkt angegeben werden.

```
public class Person {
   private String name;
   private Date geburtsdatum;
   private char geschlecht;
   private double groesse;
   private double gewicht;
}
```

Sichtbarkeiten	
UML	**Java**
-	private
+	public
#	protected
~	----------

In Java können nicht nur erbende Klassen zugreifen, sondern auch alle im selben Package

Default

Namenskonventionen in Java (naming conventions)

- Klassen, Interfaces mit großem Anfangsbuchstaben
- Primitive Datentypen, Methoden, Variablennamen mit kleinem Anfangsbuchstaben
- CamelCase für zusammengesetzte Wörter (`MeineKlasse, meineMethode`)
- Paketnamen nur Kleinbuchstaben (`meinedomain.meinepakete.paket1`)
- Konstante nur Großbuchstaben (`MEIN_KONST`)

Programmierung

- Anlegen eines Projekts
- Anlegen eines Pakets
- Erzeugen einer Java Klasse Person

```
public class Person {
  private String name;
  private Date geburtsdatum;
  private char geschlecht;
  private double groesse;
  private double gewicht;
  //...
}
```

Klasse `Date` unbekannt.
Muss aus `java.util` importiert werden.

- ...Importieren des vorbereiteten Projekts

Error vs. Warning

- **Error**: Syntaxfehler; der Text ist kein gültiges Java-Programm. Muss behoben werden, bevor Programm gestartet werden kann.
  - Z.B.: Date can not be resolved to a type
- **Warning**: Gültiges Java-Programm, aber es geschieht etwas Ungewöhnliches (wahrscheinlich nicht gewünscht).
  - Z.B.: The value of the field ... is not used

Programmierung

- Breakpoint setzen
  - "Debug" statt "Run" verwenden
  - Umschaltung in Debugperspektive erlauben
    - Resume, Suspend, Terminate, Step Into, Step Over, Step Return
- Objektinhalte ansehen
  - ... fad ...
  - obwohl: wir können z.B. einen Namen verändern. (Beachten Sie die Objekt-Ids)

Accessoren / Mutatoren

- Methoden zum Lesen / Verändern der Werte von Instanzvariablen
- Auch als getter- u. setter-Methoden bekannt (Name beginnt konventionsgemäß mit get / set).

Programmierung

- Der Eclipse Editor unterstützt die automatische Generierung von Accessoren und Mutatoren ... (Achtung: Integritätsbedingungen werden nicht automatisch überprüft!)

- ... package version2

Accessoren / Mutatoren

- Methoden zum Lesen / Verändern der Werte von Instanzvariablen
- Auch als getter- u. setter-Methoden bekannt (Name beginnt konventionsgemäß mit get / set).

Programmierung

- Der Eclipse Editor unterstützt die automatische Generierung von Accessoren und Mutatoren ... (Achtung: Integritätsbedingungen werden nicht automatisch überprüft!)

- ... package version2

Zugriff auf Methoden und Instanzvariable

- Beim Aufruf einer Methode, bzw. beim Zugriff auf eine Instanzvariable muss immer auch angegeben werden, welches Objekt verwendet werden soll.
- Syntax:
  - `objekt.methodenName()`
  - `objekt.instanzvariable`

Methodenaufrufe verketten (chaining)

- Liefert eine Methode ein Objekt zurück, kann der nächste Methodenaufruf direkt verkettet werden.
  - `objekt.methode1().methode2()` ...
  - `new GregorianCalendar().get(Calendar.YEAR)`

this

- Das Schlüsselwort `this` kann in einer beliebigen Methode verwendet werden und bezeichnet immer das "aktuelle" Objekt (also das Objekt, für welches die Methode aufgerufen wurde).
- Bei eindeutigem Zugriff auf eine Instanzvariable oder Methode des Objekts kann die explizite Angabe von `this` entfallen.
- In anderen Programmiersprachen sind auch Bezeichnungen wie `self` (z.B.: Smalltalk) oder `me` (z.B.: VisualBasic) gebräuchlich.

Deprecated

- Wörtlich übersetzt: abgelehnt
- Funktionalität, die nicht mehr den aktuellen Gepflogenheiten entspricht und wahrscheinlich in einer der nächsten Versionen entfernt wird.
- Sollte keinesfalls bei der Implementierung neuer Programme verwendet werden.

Date

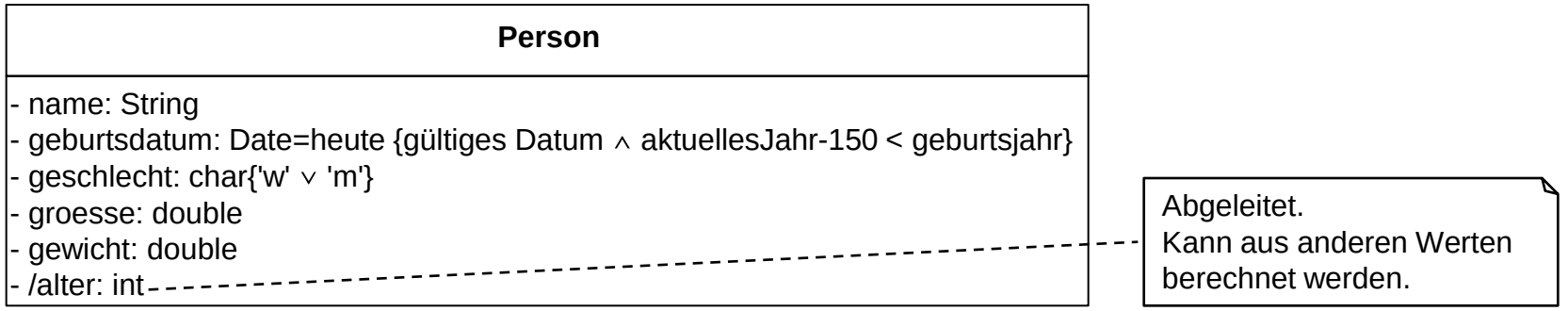
- Umgang mit Datumswerten in der Regel immer recht kompliziert (unterschiedliche Kalender, Zeitzonen, Schaltjahre, Schaltsekunden)
- Interne Darstellung oft als eine ganze Zahl
  - In java.util.date: Anzahl der Millisekunden seit 1.1.1970 00:00:00 UTC im Datentyp long
  - In unixoiden Systemen: Anzahl der Sekunden seit 1.1.1970 00:00:00 UTC
    - 19.1.2038 03:14:08 "Ende der Zeitrechnung" für 32-bit Systeme
    - 4.12.292277026596 für 64-bit Systeme (am Sonntag!)

Programmierung

- Experimente mit Date.
  - Sind Date-Objekte immer konsistent?

- package version3

Abgeleitetes Attribut (derived attribute)



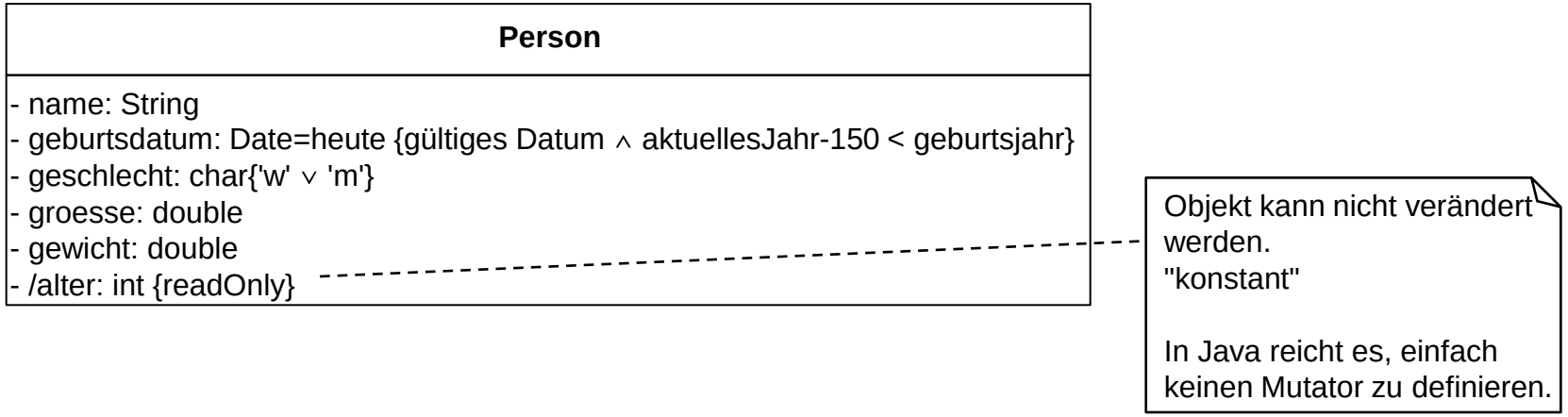

```
public class Person {
   private String name;
   private Date geburtsdatum;
   private char geschlecht;
   private double groesse;
   private double gewicht;

   public int getAlter() {
      //...
   }
}
```

Instanzvariable nicht notwendig
"Simulation" mittels Accessor.

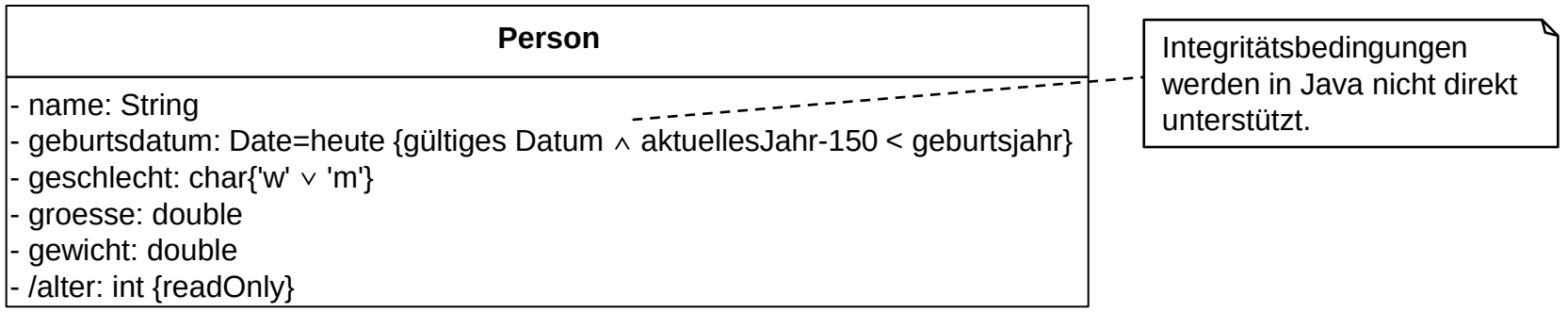
Mutator?

Mutatoren für abgeleitete Attribute

- Müssen dafür sorgen, dass die unabhängigen Variablen, die der Berechnung zugrunde liegen, entsprechend gesetzt werden.
- Oft (wie auch im konkreten Fall) nicht sinnvoll. Man verbietet dann die Änderung.



Überprüfen von Integritätsbedingungen



- typischerweise in den Mutatoren zu überprüfen:

```
public class Person {
  //…
  public void setGeburtsdatum(Date geburtsdatum) {
     GregorianCalendar datum = new GregorianCalendar();
     GregorianCalendar aktuell = new GregorianCalendar();
     datum.setTime(geburtsdatum);
     if (datum.after(aktuell) || aktuell.get(Calendar.YEAR)-
                                 datum.get(Calendar.YEAR)>150)
       throw new RuntimeException("Ungültiges Geburtsdatum");
     this.geburtsdatum = geburtsdatum;
  }
  //…
}
```

Wert wird nur gesetzt, wenn er gültig ist. Andernfalls wird das Programm abgebrochen.

Objekt bleibt konsistent!

Exceptions

- Immer, wenn im Programm ein Fehlerzustand erreicht wird, der nicht sofort und einfach behoben werden kann, sollte eine Exception ausgelöst werden.
- Das garantiert, dass Fehlerzustände nicht übersehen werden können und im weiteren Programmablauf vielleicht zu groben und eventuell teuren Fehlfunktionen des Programms führen.
- Bis auf Weiteres verwenden wir nur Exceptions vom Typ RuntimeException und Programme werden beim Werfen einer Exception mit einer lapidaren Fehlermeldung abgebrochen.

Anhang
Wiederholung Java Grundlagen

Primitive Datentypen (primitive datatypes)

Exakte Größe im Speicher nicht definiert

Typ	Werte	Größe	Default
boolean	true, false	1 Bit	false
char	Unicode Zeichen	2 Byte	\u0000
byte	Ganze Zahlen	1 Byte	(byte)0
short	Ganze Zahlen	2 Byte	(short)0
int	Ganze Zahlen	4 Byte	0
long	Ganze Zahlen	8 Byte	0L
float	"Reelle Zahlen"	4 Byte	0.0f
double	"Reelle Zahlen"	8 Byte	0.0

Berechnungen korrekt modulo 2^x

Rechenfehler
NEGATIVE_INFINITY
POSITIVE_INFINITY
NaN

Literale (literal)

- Literale sind konstante Werte, die "buchstäblich" im Programmtext auftreten.
- Boolesche Literale: `true, false`
- Ganzzahlige Literale:
  - `11, +7, -91`
  - `23l, 23L` (`int` ist default)
  - `012` (oktal)
  - `0xa1, 0Xff` (hexadezimal)
  - `0b101, 0B0101` (binär, seit J2SE 7.0)

Literale (2)

- Gleitkommazahlen Literale:
  - `27.5, +0., -.3`
  - `1.0d, 7.D, .5f, 0.12F` (`double` ist default)
  - `0.2e3, 4E-2, 0.1e+7` (technische Notation)
  - `0x1.8p2, 0XA.P-2` (hexadezimal seit JDK 5)
  - `Double.POSITIVE_INFINITY, Float.NEGATIVE_INFINITY, Double.NaN`

Jede Klasse bildet einen eigenen **Namensbereich** (namespace)

Literale (3)

- In numerischen Literalen (ganz oder Gleitkomma) kann zwischen zwei Ziffern ein _ (underscore) zum Gruppieren verwendet werden (ab Java SE 7):
  - `1_000_000`

Literale (4)

- Zeichen Literale:
  - `'a', '0'` (beliebige Unicode UTF-16 Zeichen)
  - `'\u00F1'` (Unicode Zeichennummern dürfen auch innerhalb von Zeichenketten oder in Namen von z.B. Instanzvariablen verwendet werden. Hier: ñ)
  - `'\b', '\t', '\\'` (Escape-Sequenzen; escape sequence)

Literale (5) Escape-Sequenzen

\n	newline	Zeilenvorschub
\t	tab	Tabulator
\b	backspace	Zeichen zurück
\r	carriage return	Zeilenbeginn
\f	formfeed	Seitenvorschub
\\	backslash	Umgekehrter Schrägstrich
\'	single quote	Einfaches Hochkomma
\"	double quote	Doppeltes Hochkomma
\dnnn	octal	Oktale Nummer
\xnn	hexadecimal	Hexadezimale Nummer
\unnnn	Unicode	Unicode Nummer

Literale (6)

- Zeichenketten (String) Literale:
  - `"abc","","Se\u00f1or"`
  - Strings sind in Java konstante Objekte. Das heißt: sobald das Objekt einmal erzeugt wurde, lässt sich sein Inhalt nicht mehr verändern.

Literale (7)

- Sonstige:
  - `null` (Leeres Objekt, dessen Datentyp auch `null` ist. Kompatibel mit allen Klassen)
  - `class` (Erzeugt ein Objekt für eine Klasse selbst. Z.B.: `Person.class`. Datentyp des Objekts ist `Class`. Auch für primitive Typen verwendbar; `boolean.class, void.class`)

Anhang

UML

Details zu Klassen und Objektdiagrammen

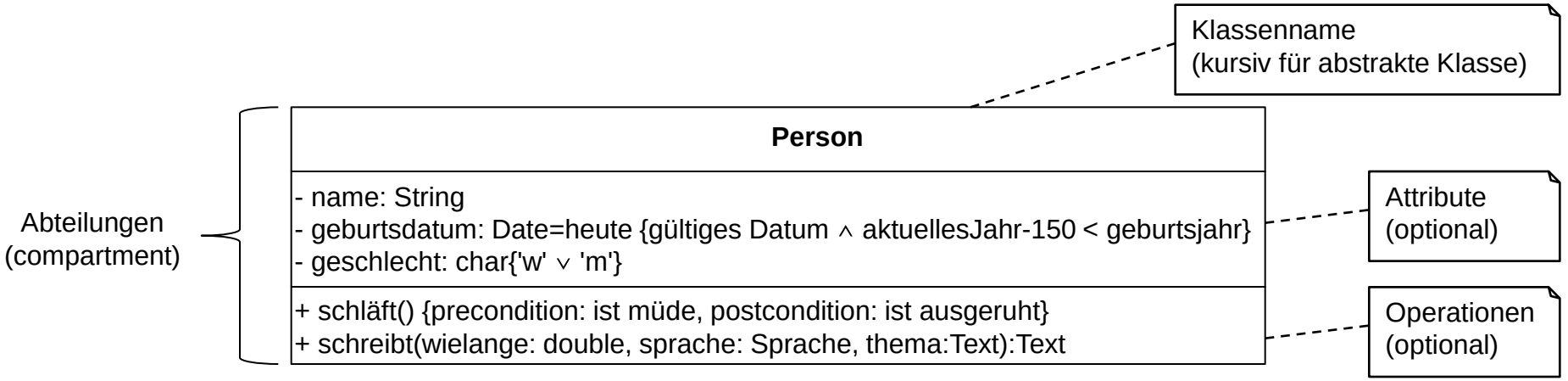
UML

- UML ist eine grafische Sprache zur objektorientierten Modellierung komplexer Systeme.
- Seit 1997 ist es ein Standard der Object Management Group (OMG).
- Die Version 2.0 wurde Ende 2004 festgelegt. Die aktuelle Version ist 2.5.1 (2.4.1 als ISO-Standard).
- UML wird laufend weiterentwickelt.
- 13 Diagrammtypen
  - statisch: Beschreibung von (möglichen) Zuständen
  - dynamisch: Beschreibung von Abläufen
- In dieser Lehrveranstaltung wird auf formale Details weitestgehend verzichtet.

Klassen- und Objektdiagramm

- statische Diagramme
- Klassendiagramm stellt alle möglichen Zustände der Objekte der Klasse dar.
- Objektdiagramm stellt einen bestimmten Zustand des Objekts (etwa zu einem definierten Zeitpunkt) dar.

UML Klassendiagramm



- Zusätzliche Compartments können bei Bedarf definiert werden.
- Viele Möglichkeiten, um Attribute und Operationen genauer zu spezifizieren. Nur die wichtigsten werden im Verlauf dieser Lehrveranstaltung behandelt.
- Nur die im jeweiligen Kontext relevanten Informationen werden dargestellt. Es können z.B. weitere Attribute oder Operationen existieren. Die in den Einführungsfolien verwendeten … sind unnötig (und streng genommen syntaktisch nicht korrekt).
- Wie kann man erkennen, ob ein Compartment Attribute oder Operationen enthält?

Attributspezifikation

- name: String
- geburtsdatum: Date=heute {gültiges Datum ∧ aktuellesJahr-150 < geburtsjahr}
- geschlecht: char{'w' ∨ 'm'}

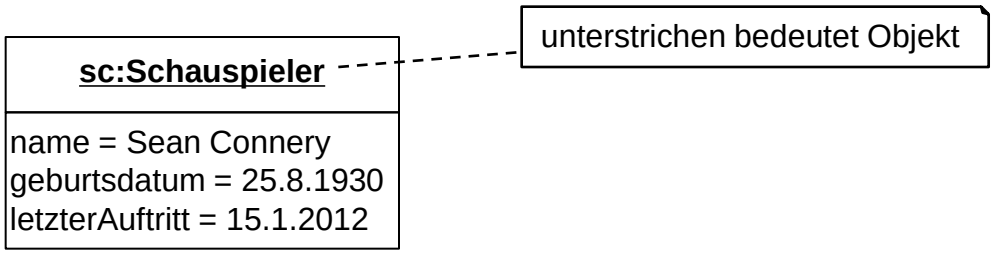
- Sichtbarkeit
- Name des Attributs
- Datentyp des Attributwerts
  - Javagrundtypen int,double,char,boolean oder beliebige Klassen
- Defaultwert
  - wird beim Erstellen eines Objekts automatisch gesetzt, falls kein anderer Wert explizit angegeben wird
- Integritätsbedingungen
  - müssen zu jedem Zeitpunkt erfüllt sein
  - in Umgangssprache oder Pseudocode
  - sehr selten auch in OCL (object constraint language)
  - können mit einem Namen versehen werden

Operationspezifikation


```
+ schläft() {precondition: ist müde, postcondition: ist ausgeruht}
+ schreibt(wielange: double , sprache: Sprache = "deutsch", thema:Text):Text
```

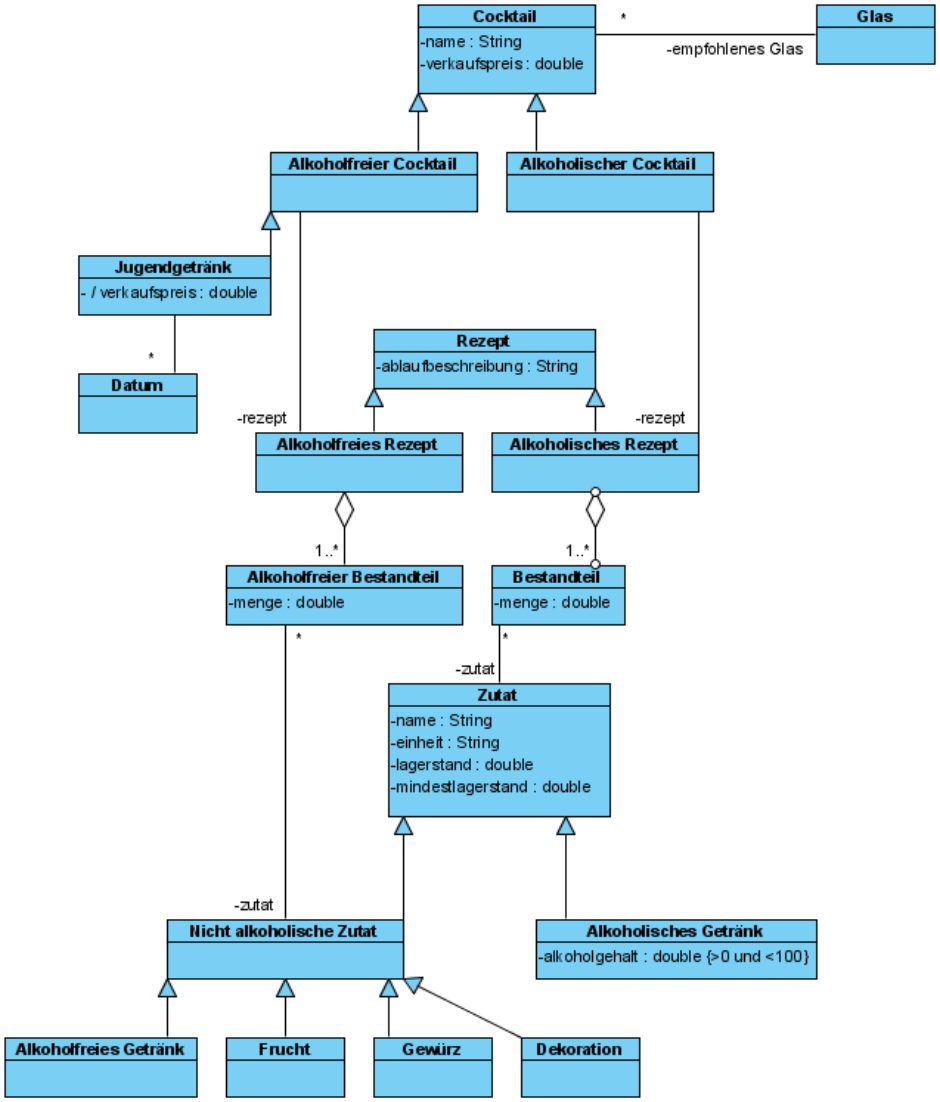
- Sichtbarkeit
- Name der Operation
  - mit Klammern für (eventuell leere) Parameterliste
- Returntyp
  - die Operation liefert als Ergebnis einen Wert diesen Typs
- Parametername
- Parametertyp
- Parameterdefault
  - Wert wird verwendet, falls beim Aufruf keiner angegeben wird
- Integritätsbedingungen
  - vordefinierte Namen precondition (muss erfüllt sein, damit die Operation ausgeführt werden kann) postcondition (muss am Ende der Operation erfüllt sein)

UML Objektdiagramm

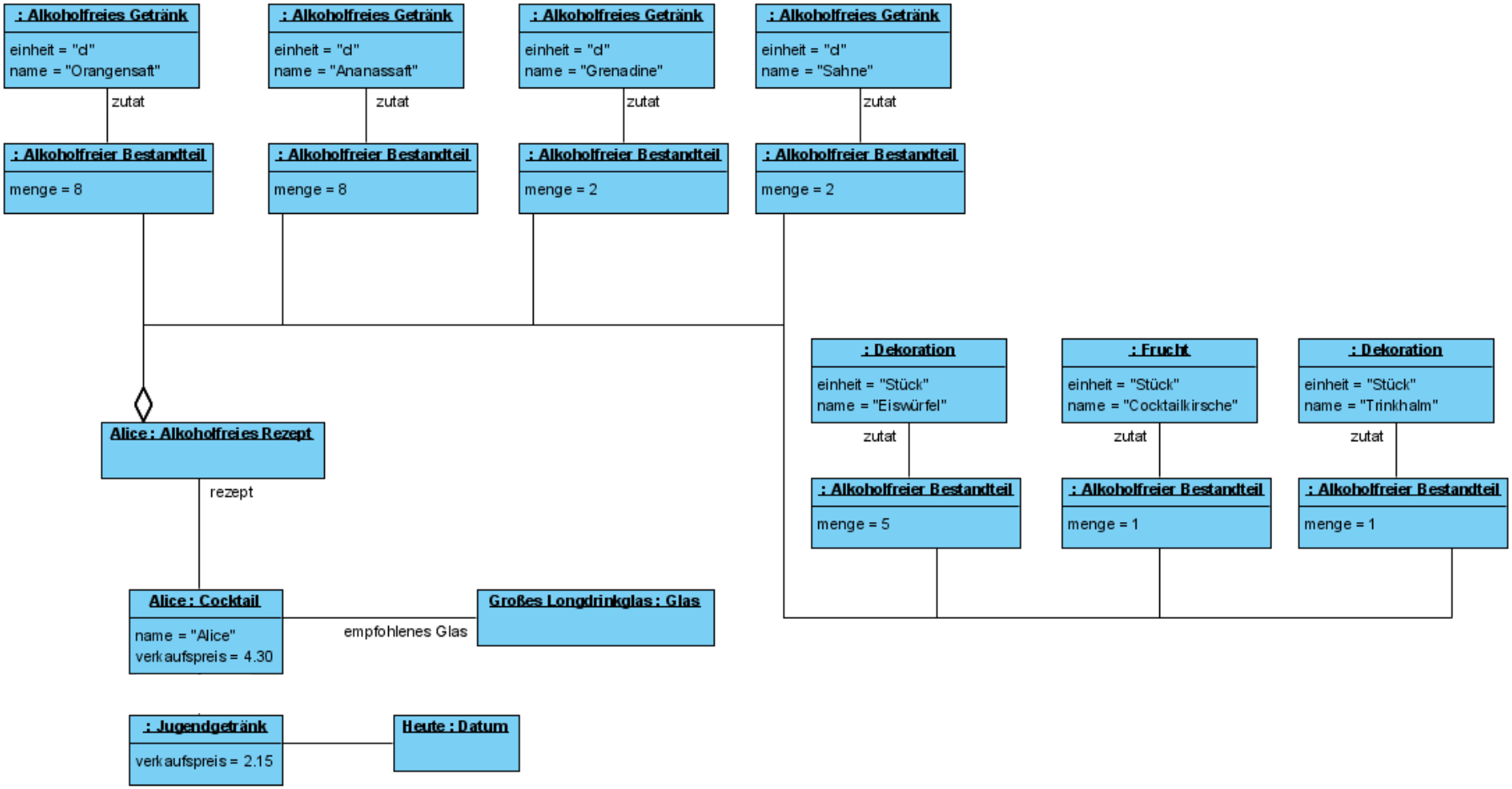


- Objektname (optional)
- :Klassenname (optional)
- Für alle Attribute (auch von Superklassen) können Werte festgelegt werden
- instanceOf ist im aktuellen Standard nicht mehr definiert (ist auch nicht nötig)

Beispiel Klassendiagramm



Beispiel Objektdiagramm



Aufgabe

- Überlegen Sie sich ein Anwendungsgebiet und modellieren Sie mindestens fünf dazugehörige Klassen.
- Entwerfen Sie ein dazu passendes Objektdiagramm
- Versuchen Sie, so viele der vorgestellten Konzepte wie möglich zu verwenden (Constraints, Vererbung, abstrakte Klassen,…)
- Implementieren Sie (mindestens) eine der Klassen in Java
- Implementieren Sie zumindest die erforderlichen Accessoren und Mutatoren (unter Berücksichtigung der Integritätsbedingungen).
- Erstellen Sie eine Funktion main(), die 10 unterschiedliche Objekte Ihrer Klasse erzeugt und die Instanzvariablen mit typischen Werten belegt.
- Stellen Sie sicher, dass Verletzungen der Integritätsbedingungen zu Exceptions führen.