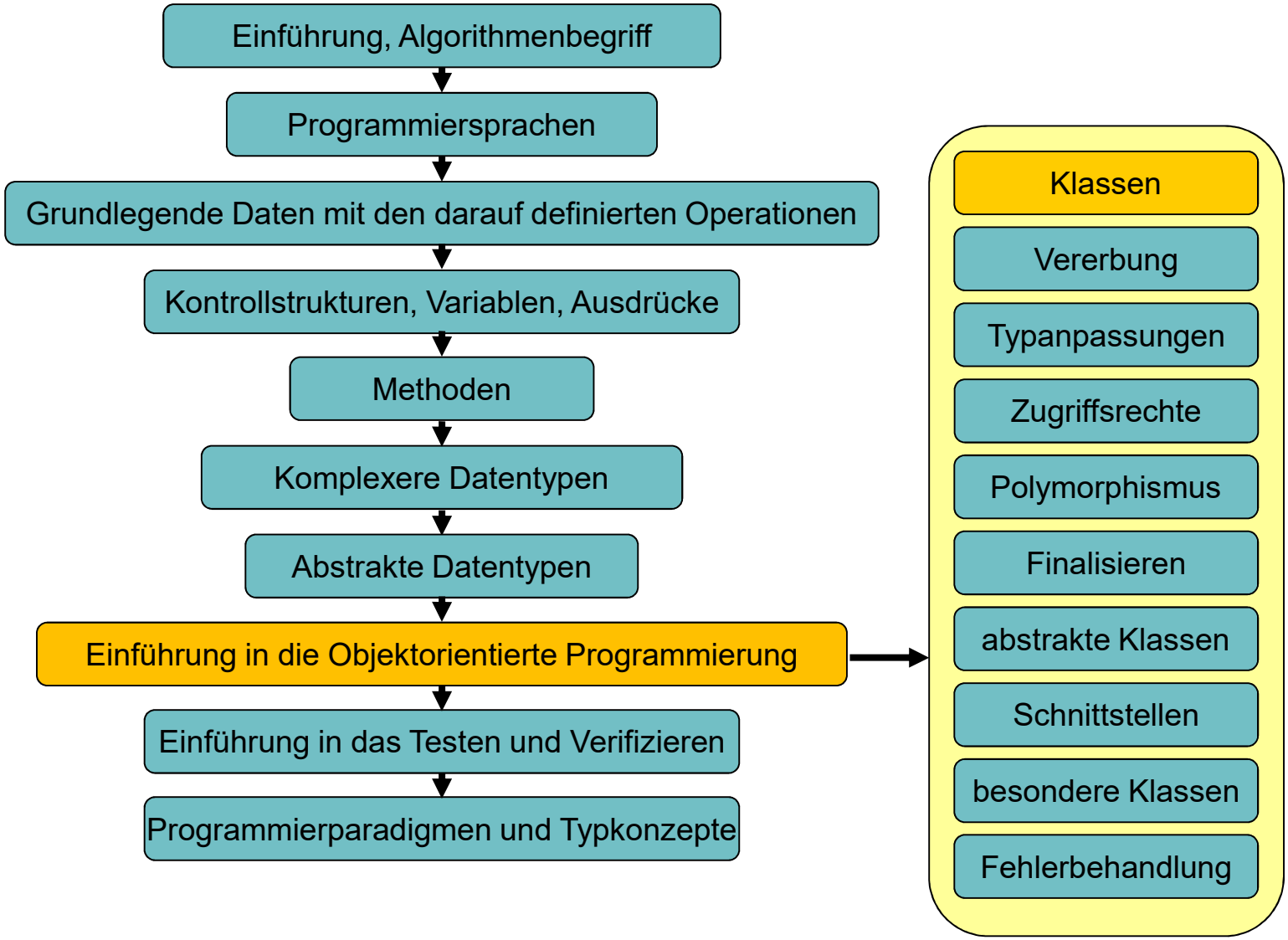
# Inhalt dieser Veranstaltung



# Nicht alles ist primitiv oder ein Vektor

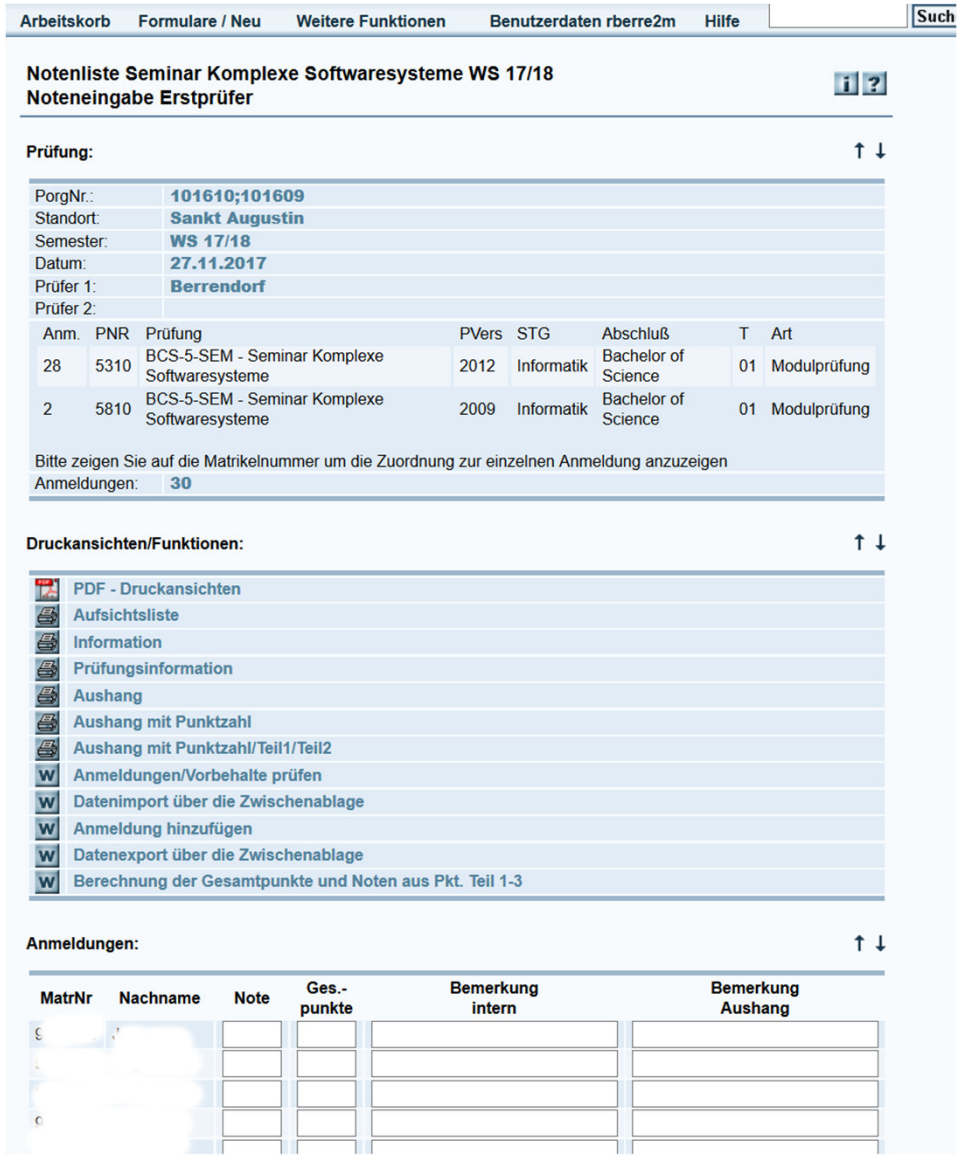
- Bis jetzt bekannt:
  - primitive Datentypen (`boolean, char, byte, short, int, long, float, double`)
  - Felder zu einem Basistyp `T`: `T[]`
  - `String`

- Was ist, wenn ich in einem Programm weder einzelne Werte, noch Felder habe, sondern strukturierte Daten wie Konten, MP3-Daten, Studierende,...? All dies bringt Java zuerst einmal **nicht** mit!

- Antwort: definiere eigene Datentypen (Klassen) basierend auf bereits bekannten Datentypen

- Zur Erinnerung: Datentyp = Wertemenge mit darauf definierten Operationen und (bei einem konkretem Datentyp) einer Kodierung der möglichen Werte

# Strukturierte Daten: Reales Beispiel



Arbeitskorb | Formulare / Neu | Weitere Funktionen | Benutzerdaten rberre2m | Hilfe | Such

**Notenliste Seminar Komplexe Softwaresysteme WS 17/18**
**Noteneingabe Erstprüfer**

**Prüfung:**

| | |
|---|---|
| PorgNr.: | 101610;101609 |
| Standort: | Sankt Augustin |
| Semester: | WS 17/18 |
| Datum: | 27.11.2017 |
| Prüfer 1: | Berrendorf |
| Prüfer 2: | |

| Anm. | PNR | Prüfung | PVers | STG | Abschluß | T | Art |
|---|---|---|---|---|---|---|---|
| 28 | 5310 | BCS-5-SEM - Seminar Komplexe Softwaresysteme | 2012 | Informatik | Bachelor of Science | 01 | Modulprüfung |
| 2 | 5810 | BCS-5-SEM - Seminar Komplexe Softwaresysteme | 2009 | Informatik | Bachelor of Science | 01 | Modulprüfung |

Bitte zeigen Sie auf die Matrikelnummer um die Zuordnung zur einzelnen Anmeldung anzuzeigen

Anmeldungen: 30

**Druckansichten/Funktionen:**

- PDF - Druckansichten
- Aufsichtsliste
- Information
- Prüfungsinformation
- Aushang
- Aushang mit Punktzahl
- Aushang mit Punktzahl/Teil1/Teil2
- Anmeldungen/Vorbehalte prüfen
- Datenimport über die Zwischenablage
- Anmeldung hinzufügen
- Datenexport über die Zwischenablage
- Berechnung der Gesamtpunkte und Noten aus Pkt. Teil 1-3

**Anmeldungen:**

| MatrNr | Nachname | Note | Ges.-punkte | Bemerkung intern | Bemerkung Aushang |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# Beispiel zu strukturierten Daten

- Die Leistung einer/eines beliebigen Studierenden in einer Klausur kann beschrieben werden durch insgesamt diese Informationen (in Klammern ein möglicher Java-Typ für die jeweiligen Teilinformationen)
  - eine Matrikelnummer (`int`)
  - Vorname (`String`)
  - Nachname (`String`)
  - Note (`double`)
  - Punktzahl (`int`)

| abstrakt |
|---|
| matrikelnummer : int |
| vorname : String |
| nachname : String |
| note : double |
| punktzahl : int |

Das **Formular** sieht für **alle** Prüfungsleistungen gleich aus!

- Zwei **konkrete** Prüfungsleistungen würden dann zum Beispiel so angegeben:

| abstrakt | konkret |
|---|---|
| matrikelnummer : int | 76534321 |
| vorname : String | Susi |
| nachname : String | Schlau |
| note : double | 2,0 |
| punktzahl : int | 91 |

| abstrakt | konkret |
|---|---|
| matrikelnummer : int | 1234567 |
| vorname : String | Ludwig |
| nachname : String | Lustig |
| note : double | 4,0 |
| punktzahl : int | 63 |

spezifisch für diese Prüfungsleistung

# Etwas anderes: wir simulieren einen Zoo



Bilderquelle: Kölner Zoo

- Aufgabenstellung: wir modellieren einen Zoo in Software.
- In einem Zoo gibt es Tiere, die in Gehegen leben
- An einem Gehege vorne angebracht ist eine Nummer zur Orientierung der Besucher sowie die aktuelle Anzahl der Bewohner. Jedes Gehege hat eine maximale Belegungszahl an Tieren.
- Jedes Tier hat ein eindeutige Nummer. Manche Tiere haben auch einen Namen. Tiere leben in einem Gehege. Tiere können zwischen Gehegen umziehen. Patentiere sind Tiere mit einem Einkommen.

# Und nun die Modellierung in Software...

- Frage: wie kann ich 800 Tiere und 45 Gehege sinnvoll in einem Programm darstellen und damit arbeiten? 800+45 Programme schreiben???
- Teilantwort: Klassenbildung gleichartiger Dinge

- Alle Tiere (und alle Gehege) sind in dem Beispiel aus abstrakter Sicht sehr ähnlich
- Alle haben im Rahmen dieser Aufgabenstellung gleiche abstrakte Eigenschaften (Nummer, Name, leben in einem Gehege,...) und Fähigkeiten (können umziehen)

- Deshalb:
  - Schritt 1: gebe eine allgemeine abstrakte Beschreibung eines Tieres an (Beschreibe eine Klasse `Tier`)
  - Schritt 2: jedes konkrete Tier lässt sich dann basierend auf dieser abstrakten Beschreibung erzeugen und spezifisch behandeln

# Übersicht über die nächsten Minuten ...

- Klassen dienen der einmaligen abstrakten Beschreibung gleichartiger Dinge
- Aus einer Klasse kann ich beliebig viele Objekte dieser Art erzeugen und damit arbeiten
- In einer Referenzvariablen kann ich einen Verweis auf solch ein Objekt speichern und darüber Bezug nehmen auf dieses Objekt

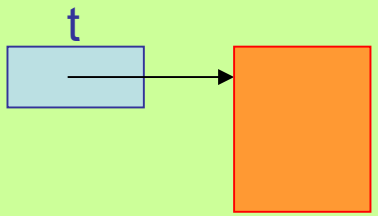
- Achtung: Nicht alle Details werden hier immer angegeben. Eine Nacharbeitung mit Hilfe des Skripts ist nötig (und wird so vorausgesetzt)!

abstrakte Beschreibung

```
public class Tier {
    ...
}
```

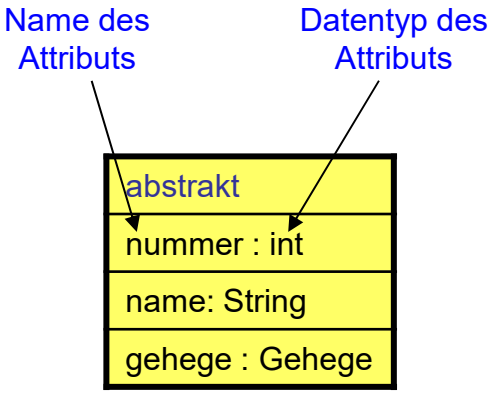
konkrete Nutzung

```
Tier t = new Tier();
```

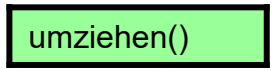


# Klassenbildung

- Jedes Tier hat gewisse Eigenschaften oder **Attribute**:
  - eindeutige Nummer
  - Name (evtl. nicht vorhanden)
  - lebt in Gehege (bei Geburt noch nicht vorhanden)



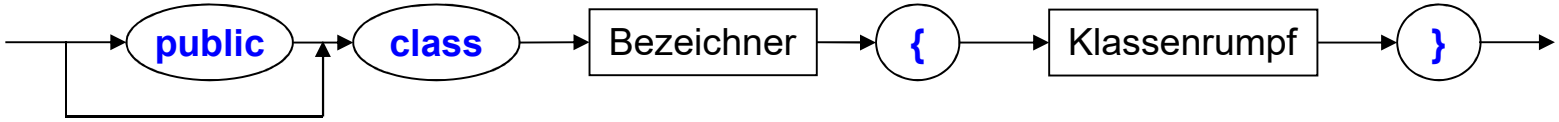

- Jedes Tier hat gewisse **Fähigkeiten / Aktivitäten**:
  - Geburt
  - Tod
  - Umzug zwischen Gehegen



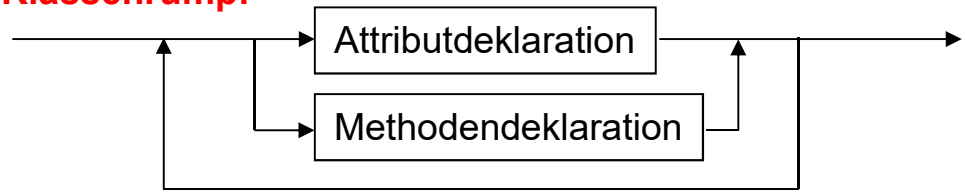

- Wir können also eine allgemeine, aber zweckbestimmte Beschreibung für eine Klasse gleichartiger Dinge angeben, die abstrakt für alle diese gleichartigen Dinge gilt.
- Konkrete Werte für Tiername etc. werden bei der Klassenbildung weggelassen, nur **dass** es einen Tiernamen gibt ist für eine Klasse relevant (gelber abstrakter Teil im Formular)!

# Java Klassendefinition

**Klassendefinition**




**Klassenrumpf**




- Durch eine Klasse wird in Java ein neuer Datentyp definiert
- Der Name der Klasse kann überall dort verwendet werden, wo ein Typ verlangt ist (z.B. Variablendeklaration, Ergebnistyp einer Methode,...)

```
public class Tier {

    // zuerst werden üblicherweise die Attribute angegeben

    // anschliessend werden die Methoden angegeben

}
```

# Attribute / Instanzvariablen

- Ein Attribut beschreibt eine Eigenschaft einer Klasse gleichartiger Objekte
- Wie finde ich die? Attribute einer Klasse leiten sich oft ab aus **Nomen / Hauptwörtern** der umgangssprachlichen Aufgabenbeschreibung (Nummer, Name, ...)

- Attribute werden in Java ähnlich wie Variablen genutzt (gleich mehr dazu)

- Attribute, die eine Eigenschaft **eines** Objektes beschreiben, nennt man Instanzvariablen

- Instanzvariablen eines primitiven Typs lassen sich direkt mit dem entsprechenden vordefinierten Typ in der Deklaration angeben.
- Falls kein primitiver Typ vorliegt, muss es ein Referenztyp sein (Beispiel `Gehege`). Die damit deklarierte Variable ist eine Referenzvariable (Details später).

# Beispiel

```
public class Tier {

    // Instanzvariablen
    int nummer;             // eindeutige Nummer des Tieres
    String name;            // Name des Tieres (evtl. nicht vorhanden)
    Gehege gehege;          // Gehege, in dem es wohnt

    // anschliessend werden die Methoden angegeben
}
```

```
public class Gehege {

    // Instanzvariablen
    int nummer;               // eindeutige Nummer
    int maximalBelegung;      // max. Anzahl an Tieren in Gehege
    int aktuelleBelegung;     // derzeitige Anzahl
    Tier[] bewohner;          // derzeitige Tiere

    // anschliessend werden die Methoden angegeben
}
```

# Methoden

- Methoden geben Aktivitäten oder Fähigkeiten an, die ein Objekt dieser Klasse ausführt oder kann
- Wie finde ich die? Sie ergeben sich oft aus **Verben** der umgangssprachlichen Beschreibung (umziehen,...)

- Methoden in diesem Zusammenhang werden **ohne** `static` angegeben (zu Methoden mit `static` kommen wir aber später wieder zurück)

- Man nennt diese Methoden Instanzmethoden

- Instanzmethoden sind immer im Zusammenhang mit einem Bezugsobjekt zu sehen und werden immer mit einem Bezugsobjekt genutzt (aufgerufen)
- Die Nutzung von Instanzvariablen in einer Instanzmethode bezieht sich dann immer auf dieses konkrete Bezugsobjekt

# Beispiel Tier

In dieser Instanzvariablen steht der Name des konkreten Tieres

liefere Namen des konkreten Tieres

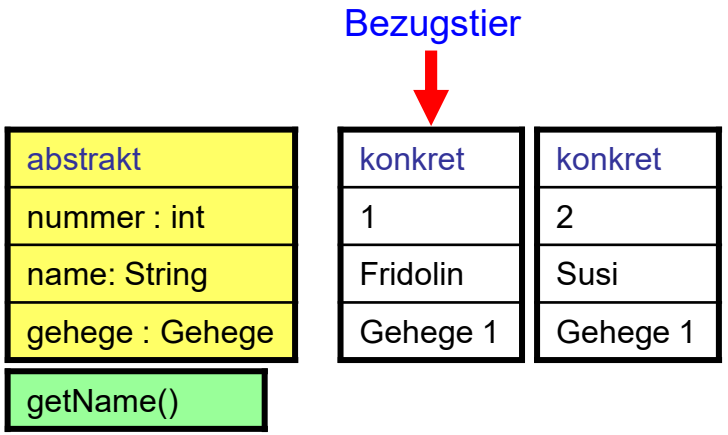
```
public class Tier {

    // Instanzvariablen
    int nummer;             // eindeutige Nummer des Tieres
    String name;            // Name des Tieres
    Gehege gehege;          // Gehege, in dem es wohnt

    // Instanzmethoden

    // liefere Name des Tieres
    public String getName() {
        return name;
    }

    // aendere Name des Tieres
    public void setName(String neuerName) {
        name = neuerName;
    }
}
```

| abstrakt | konkret |
|---|---|
| matrikelnummer : int | 76534321 |
| vorname : String | Susi |
| nachname : String | Schlau |
| note : double | 2,0 |
| punktzahl : int | 91 |

# Bezug bei Instanzmethode/-variable

## Ausschnitt Tier-Klasse

```
public class Tier {

    // Instanzvariablen
    String name;           // Name des Tieres

    // liefere Name des Tieres
    public String getName() {
        return name;
    }
}
```




Die Tierklasse beschreibt abstrakt jedes Tier mit diesen Eigenschaften. Genutzt werden die Instanzmethoden aber immer im Zusammenhang mit einem konkreten Tierobjekt.

# Zwischenstand

- Beliebig viele Daten gleicher Struktur lassen sich einmalig abstrakt beschreiben über die gemeinsamen Eigenschaften und Fähigkeiten (Klassenbildung)
- Eine Java-Klasse enthält Attribute (beschreiben Eigenschaften) und Methoden (beschreiben Fähigkeiten)
- Instanzvariablen und –methoden beziehen sich immer auf ein Bezugsobjekt

**Reflektion**

- Geben Sie in einer Schemazeichnung (analog zu meinen Zeichnungen mit Studenten) zwei beliebige konkrete Objekte zur folgenden Klasse an:

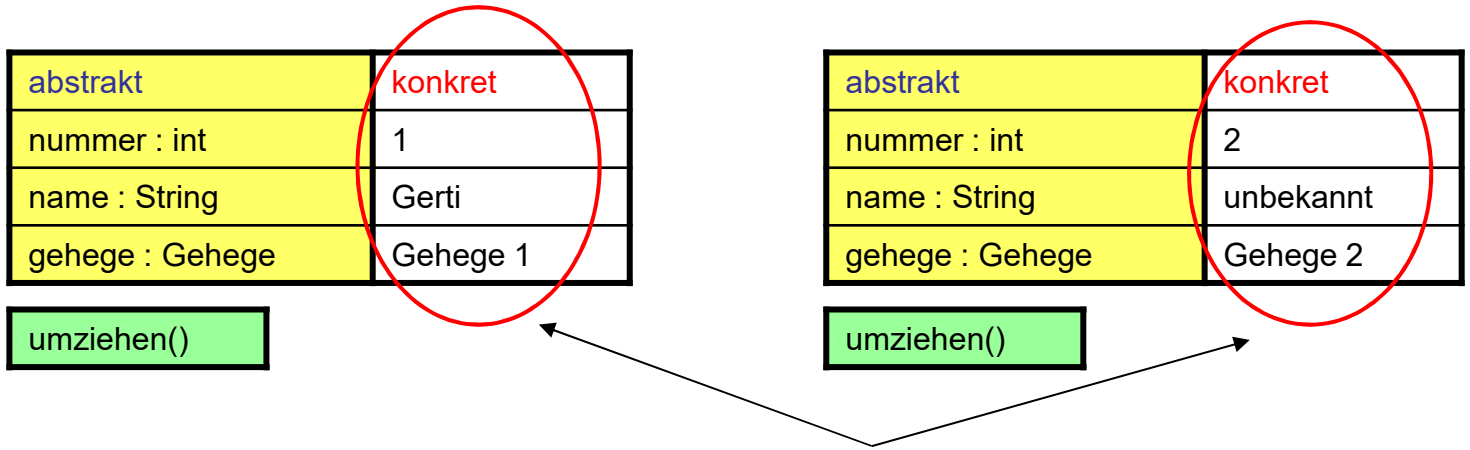
```
public class Test {
   int a,b; float x;
   int getA() { return a; }
}
```

# Beliebig viele Objekte basierend auf einer Klasse

- Liegt eine Klassendefinition vor, so lassen sich beliebig viele konkrete Objekte (hier Tiere) dann basierend auf dieser allgemeinen Beschreibung angeben

- Ein konkretes Objekt nennt man Instanz der Klasse, den Vorgang des Erzeugens eines Objekts aus einer Klasse Instanziierung.

- Beispiel hier: 2 Tiere aus Tier“formular“ (Klasse `Tier`) erzeugt




auszufüllen für jedes konkrete Tier,
beschreibt den Zustand eines Tieres vollständig

# Zustand

- Alle Instanzen haben die gleichen Instanzvariablen (gelber Teil des Formulars), jede Instanz hat aber eigene **Werte** für die Attribute / Instanzvariablen (weißer Teil des Formulars)
- Die konkreten Werte aller Instanzvariablen eines Objekts beschreiben den aktuellen Zustand, in dem sich **dieses** Objekt befindet
- Durch Änderung einer Instanzvariablen ändert sich auch der Zustand des Objekts

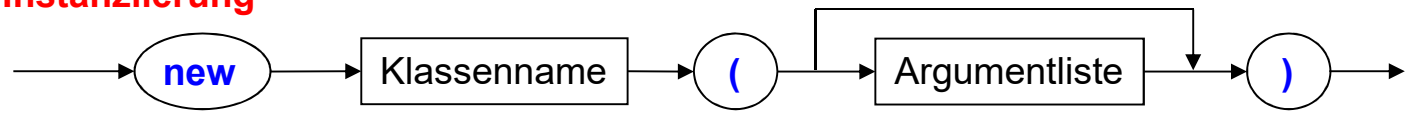
- Beispiel: ein Tier heißt Gerti, hat die Nummer 1 und lebt derzeit in Gehege 1

| abstrakt | konkret |
|---|---|
| nummer : int | 1 |
| name : String | Gerti |
| gehege : Gehege | Gehege 1 |
| umziehen() | |

| abstrakt | konkret |
|---|---|
| nummer : int | 2 |
| name : String | unbekannt |
| gehege : Gehege | Gehege 2 |
| umziehen() | |

# Instanziierung in Java

**Instanziierung**




- Ähnlich wie dies bei Feldern geschah, kann man auch bei Klassen mit dem new-Operator ein neues Objekt zu einer Klasse erzeugen
- Das Ergebnis des new-Operators ist eine Referenz auf das neu geschaffene Objekt

```
new Tier()
```

| abstrakt | konkret |
|---|---|
| nummer : int | 1 |
| name : String | unbekannt |
| gehege : Gehege | null |

umziehen()

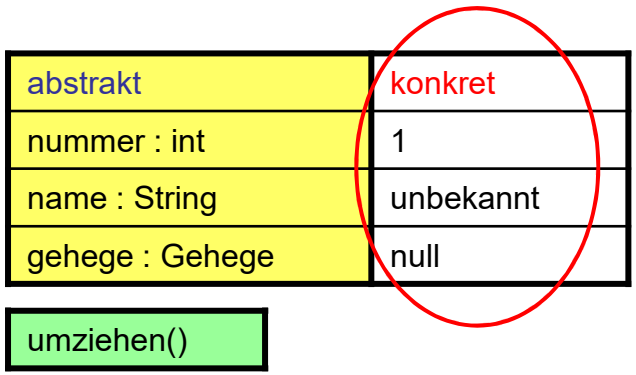
nur der Zustand des Objekts
muss auf dem Heap gespeichert werden
(weißer Teil des Formulars)

# Details der Instanziierung

1. Der new-Operator besorgt sich für das neue Objekt Speicher auf dem Heap
2. Alle Instanzvariablen des neuen Objekts werden mit dem Null-Wert des entsprechenden Datentyps initialisiert
3. Der durch die Argumenttypen in der Argumentliste im new-Aufruf spezifizierte Konstruktor wird aufgerufen (gleich mehr zu Konstruktoren)
4. Als Resultat des new-Operators wird die Referenz auf das neu geschaffene Objekt geliefert

Die Lebensdauer eines Objekts ist von der Instanziierung bis zu dem Zeitpunkt, wo es keine aktive Referenz mehr auf das Objekt existiert
(→ Garbage Collector)

```
new Tier()
```




nur der Zustand des Objekts
muss auf dem Heap gespeichert werden

# Zwischenstand

- Aus einer Klassenbeschreibung lassen sich beliebig viele konkrete Instanzen / Objekte mit dem `new` Operator erzeugen
- Zu jeder Instanz wird der aktuelle Zustand (Inhalt aller Instanzvariablen) auf dem Heap gespeichert

**Reflektion**

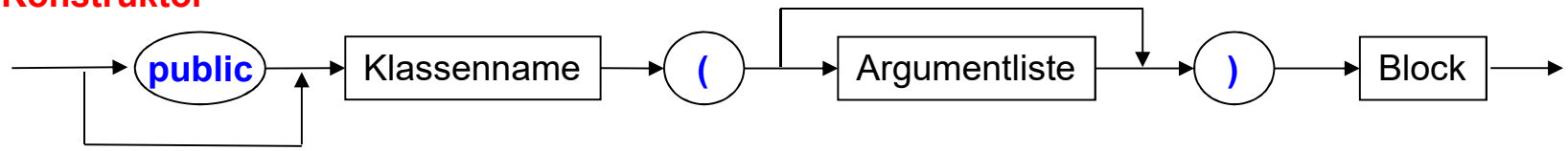
- Gegeben ist folgende Klasse: `public class Test { int a,b; }`
- Wie gibt man die Erzeugung eines Objekts dieser Klasse in Java an?
- Zeichen Sie sich ein Diagramm, wie dieses geschaffene Objekt aussieht.

# Konstruktor

- Es gibt spezielle Methoden, die bei Erzeugung eines Objekts implizit oder explizit aufgerufen werden
- Solche Methoden heißen Konstruktoren
- In einem Konstruktor wird üblicherweise dem neu erzeugten Objekt ein definierter Startzustand gegeben
- Beispiel: Instanzvariablen auf bestimmte Werte gesetzt, die ungleich 0 sind

# Angabe eines Konstruktors

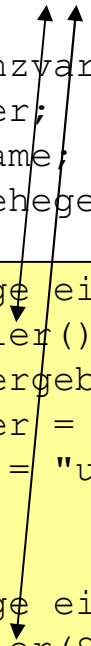
**Konstruktor**




- Ein Konstruktor ist syntaktisch gesehen ähnlich einer Methode aufgebaut
- Unterschiede:
  - der Name des Konstruktors entspricht dem Klassennamen (inkl. vereinbarungsgemäß Großbuchstaben zu Beginn)
  - der Ergebnistyp ist nicht vorhanden
- Analog zu Methoden kann es mehrere Konstruktoren geben (der Name wird also überladen), die sich in ihrer Signatur unterscheiden müssen
- Gibt man selbst keinen Konstruktor an (und nur dann!), so ist implizit ein parameterloser Konstruktor (Default-Konstruktor) definiert mit leerem Block. Dies wird später **wichtig** (Stichwort Vererbung).

# Beispiel

```
public class Tier {

    // Instanzvariablen
    int nummer;             // eindeutige Nummer des Tieres
    String name;            // Name des Tieres (evtl. nicht vorhanden)
    Gehege gehege;          // Gehege, in dem es wohnt

    // Erzeuge ein neues Tier ohne Namen
    public Tier() {
        // vergebe automatisch eine neue Nummer. Wie???
        nummer = 1;
        name = "unbekannt";
    }

    // Erzeuge ein neues Tier mit Namen
    public Tier(String tierName) {
        // vergebe automatisch eine neue Nummer. Wie???
        nummer = 1;
        name = tierName;
    }

   ...
}
```

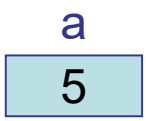


Zwei Konstruktoren vorhanden, einer ist parameterlos, der andere mit einem Parameter.

# Wiederholung Variablen

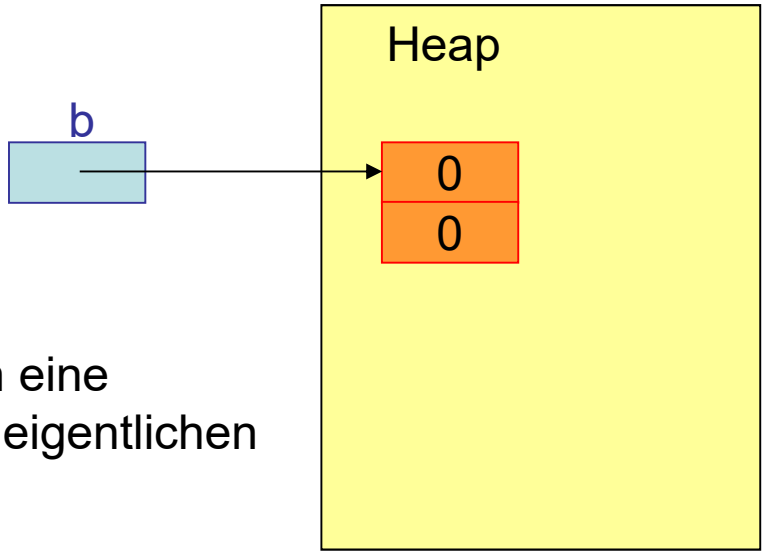
- primitiver Typ

```
int a = 5;
```




- Feld (ein Referenztyp)

```
int[] b = new int[2];
```




- Variablen zu einem Referenztyp enthalten eine Referenz (Hauptspeicheradresse) zu den eigentlichen Daten (Feldinhalt)

# Klassen sind auch Referenztypen

- primitiver Typ

```
int a = 5;
```

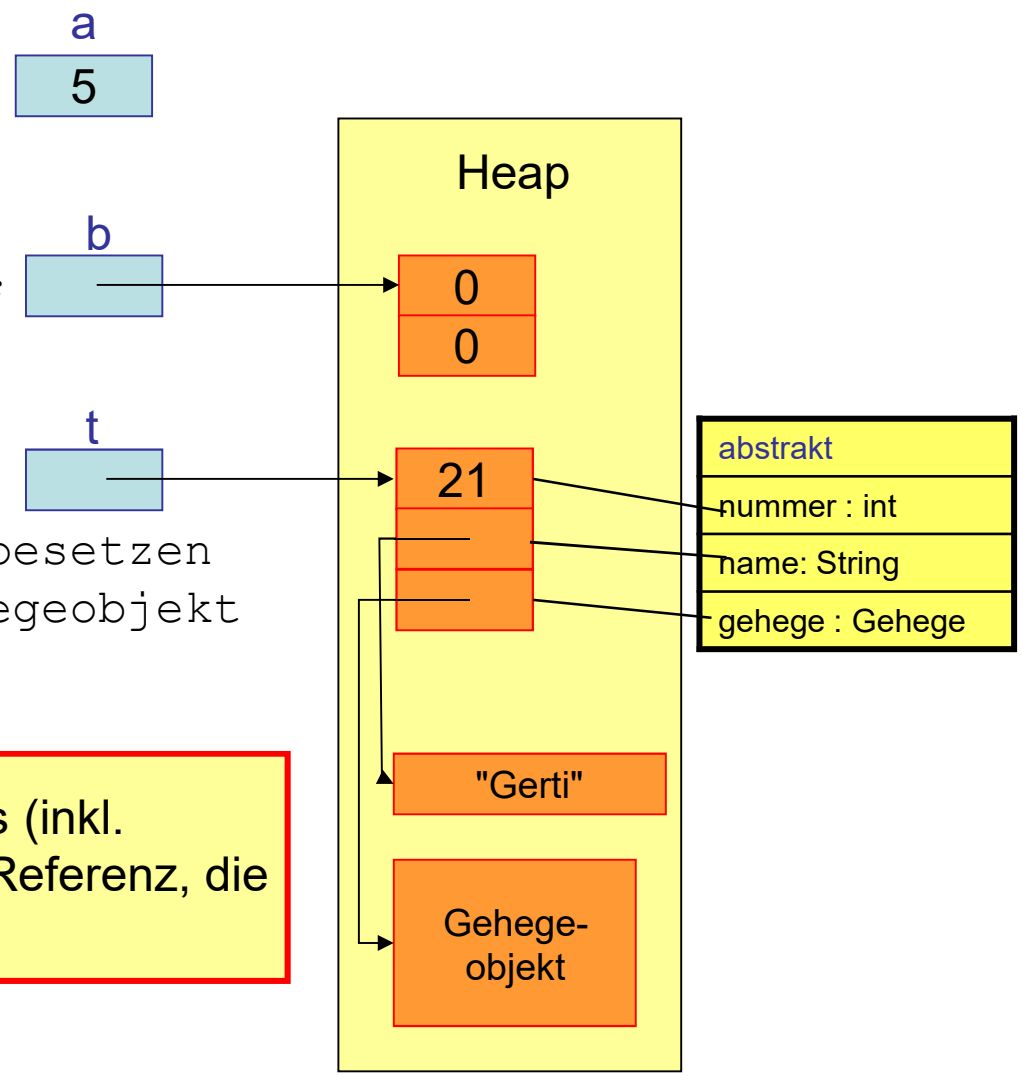
- Feld (Referenztyp)

```
int[] b = new int[2];
```

- Klasse (Referenztyp)

```
Tier t = new Tier();
// Instanzvariablen besetzen
// mit "Gerti" / Gehegeobjekt
```

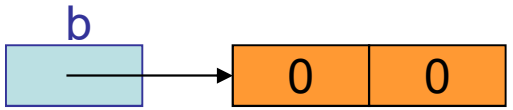
- Alle Variablen eines Referenztyps (inkl. Instanzvariablen) enthalten eine Referenz, die auch `null` sein kann



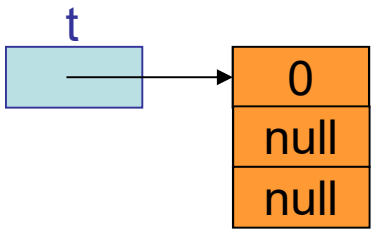
# Referenzvariablen und Instanziierung

Muster: `Datentyp Referenzvariable = Objekt;`

```
// Feld
int[] b = new int[2];
```



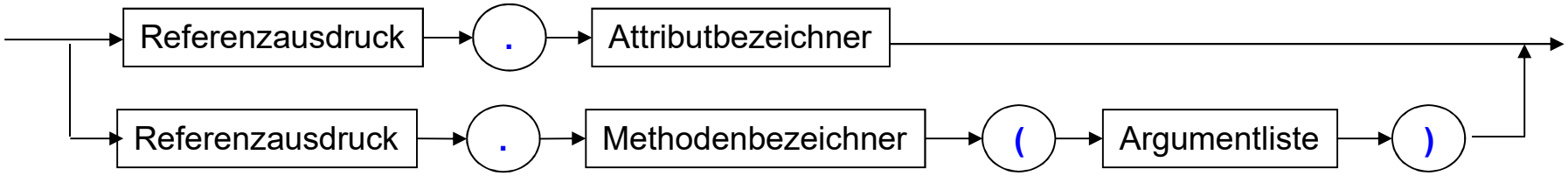

```
// Klasse
Tier t = new Tier();
```




- Mit dem new-Operator wird ein anonymes Objekt erzeugt (also ohne Namen)
- Das Ergebnis des Operators ist eine Referenz auf dieses neue Objekt
- Das Objekt ist nur über diese Referenz erreichbar
- In einer entsprechend typisierten Referenzvariablen kann/sollte man die Referenz auf das Objekt speichern und kann so auch später auf das Objekt zugreifen

# Nutzung der Funktionalität eines Objektes

**Ausdruck (Erweiterung des bisherigen Syntaxdiagramms)**




- Ein Referenzausdruck ist ein Ausdruck, dessen Ergebnis eine Referenz ist
- Normalfall: Variable eines Referenztyps

- Über einen solchen Referenzausdruck kann man auf die Instanzvariablen des referenzierten Objekts zugreifen und die Methoden dieses Objekts nutzen
- Der Referenzausdruck legt also das Bezugsobjekt fest, zu dem relativ das Attribut oder die Methode genommen werden soll

# Beispiel zur Nutzung

```
public class Zoo {

    public static void main(String[] args) {

        Tier t1 = new Tier();
        Tier t2 = new Tier("Gerti");

        // Standardgehege
        Gehege g1 = new Gehege();
        // Grossgehege mit maximal 10 Tieren
        Gehege g2 = new Gehege(10);

        // Namen von Tier 1 und 2 besorgen
        String s1 = t1.getName();
        String s2 = t2.getName();
    }
}
```

Objekte erzeugen über Konstruktoren

über Referenz Funktionalität des Objekts abrufen

# Weiteres zusammenhängendes Beispiel

- Aufgabe: modelliere in Software Punkte und Linien im zweidimensionalen Raum

- Punkte
  - haben reelle x- und y-Koordinate
  - Koordinatenangaben lassen sich erfragen
  - lassen sich verschieben
  - können auf dem Bildschirm gemalt werden
  - lassen sich als Text auf dem Bildschirm ausgeben

- Linien
  - besitzen Start- und Endpunkt
  - lassen sich verschieben. Start- und Endpunkt werden dabei beide um den gleichen Betrag verschoben.
  - können auf dem Bildschirm gemalt werden
  - lassen sich als Text auf dem Bildschirm ausgeben

# Klasse Punkt

```
public class Punkt {

    // Koordinaten des Punktes
    private double x;
    private double y;

    // Konstruktor
    public Punkt(double x1, double y1) {
        x = x1;
        y = y1;
    }

    // liefere x-Koordinate
    public double getX() {
        return x;
    }

    // setze x-Koordinate
    public void setX(double neuerWert) {
        x = neuerWert;
    }

    // y-Koordinate analog

...
```

```
...
    // verschiebe den Punkt
    public void verschieben(double dx, double dy) {
        x += dx;
        y += dy;
    }

    // male den Punkt
    public void malen(Zeichenblatt zb) {
        zb.punkt(getX(), getY());
        zb.anzeigen();
    }

    // liefere Textdarstellung
    public String toString() {
        return "(" + getX() + ","
                   + getY() + ")";
    }
}
```

# Klasse Linie

```
public class Linie {

    // Start- und Endpunkt
    private Punkt p1;
    private Punkt p2;

    // erzeuge eine Linie
    public Linie(double x1, double y1,
                 double x2, double y2) {
        p1 = new Punkt(x1, y1);
        p2 = new Punkt(x2, y2);
    }

    // erzeuge eine Linie
    public Linie(Punkt q1, Punkt q2) {
        p1 = new Punkt(q1.getX(), q1.getY());
        p2 = new Punkt(q2.getX(), q2.getY());
    }

    // verschiebe die Linie
    public void verschieben(double dx, double dy) {
        p1.verschieben(dx, dy);
        p2.verschieben(dx, dy);
    }
...
```

```
...
    // male die Linie
    public void malen(Zeichenblatt zb) {
        zb.gehe(p1.getX(), p1.getY());
        zb.linie(p2.getX(), p2.getY());
        zb.anzeigen();
    }

    // liefere eine Textdarstellung
    public String toString() {
        return p1.toString()
                 + "-" + p2.toString();
    }
}
```

Frage: wieso steht dort nicht

```
p1 = q1;
p2 = q2;
```

Wir haben doch 2 Punkte `p1,p2`?

# Nutzung der beiden Klassen

```
public class Test {

    public static void main(String[] args) {

        // 2 Punkte erzeugen
        Punkt p1 = new Punkt(25, 25);
        p1.malen(zb);
        Punkt p2 = new Punkt(100, 100);
        p2.malen(zb);

        // eine Linie erzeugen
        Linie l = new Linie(p1, p2);
        l.malen(zb);

        // Linie verschieben
        l.verschieben(10.0, 12.0);
        l.malen(zb);
    }
}
```

Hinweis: einige Details zum Malen hier weggelassen (vollständiger Code auf Webseite)

# Klassenvariablen

- Eine Instanzvariable existiert für jedes instanziierte Objekt ein mal
- Eine solche Variable speichert einen Teil des Objektzustands eines Objekts

- Was ist mit Informationen, die nicht den Objektzustand beschreiben, aber zu der Klasse gehören (Datenkapselung)?
- Beispiel: Nachhalten der Anzahl aller erzeugten Tiere

- Antwort: Klassenvariablen
- Unterschied in Definition zu Instanzvariablen: zusätzlich `static` in der Deklaration angeben
- Bedeutung: diese Klassenvariable existiert genau ein mal für diese Klasse und ist damit unabhängig von Objekten
- Anwendung: zählen von irgendwas, globaler Zustand (im Gegensatz zu Objektzustand)
- Klassenvariablen werden im statischen Datenbereich angelegt und haben eine Lebensdauer vom Start des Programms bis zum Ende

# Klassenmethoden

- Analog zur Unterscheidung Instanz-/Klassenvariablen: jetzt auch Instanz- / Klassenmethoden

- Klassenmethode: zusätzlich `static` angeben

- Mit Klassen gelernt (Instanzmethode):

```
public double xyz();
```

- Von früher schon bekannt (jetzt klar: Klassenmethode):

```
public static double xyz();
```

- Bedeutung Klassenmethode: diese Methode kann immer aufgerufen werden (auch ohne Bezugsobjekt).

# Beispiel

```
public class Tier {
    // Klassenvariablen
    private static int neueNummer = 0;  // Zaehler fuer alle Tiere

    // Instanzvariablen
    private int nummer;             // eindeutige Nummer des Tieres
    private String name;            // Name des Tieres (evtl. nicht vorhanden)
    private Gehege gehege;          // Gehege, in dem es wohnt

    /**
     * Erzeuge ein neues Tier ohne Namen
     */
    public Tier() {
        // vergebe automatisch eine neue Nummer
        nummer = ++neueNummer;
        name = "unbekannt";
        // gehege ist bereits null
    }

    // Klassenmethode: liefere Anzahl bis jetzt erzeugter Tiere
    public static int anzahlTiere() {
        return neueNummer;
    }
}
```

# Gegenüberstellung Klassen-/Instanzvariablen

| | Klassenvariable | Instanzvariable |
|---|---|---|
| Beispiel | `public class Test {`<br>`  static int x;`<br>… | `public class Test {`<br>`  int x;`<br>… |
| Existenz | genau ein mal | für jedes Objekt ein mal |
| Lebensdauer | Laufzeit des Programms | Lebensdauer des Objekts |
| Zugriff | innerhalb der Klasse mit einfachem Namen<br>oder über Klassenname<br>oder über Referenz (letzten beiden Zugriffsrechte beachten) | innerhalb der Klasse mit einfachem Namen<br>oder über Referenz (dann Zugriffsrechte beachten) |

Der Zugriff auf Instanzmethoden / Klassenmethoden ist analog (Beispiel gleich):

- Instanzmethode: innerhalb der Klasse nur innerhalb einer Instanzmethode (wieso?) über einfachen Namen oder (auch außerhalb der Klasse) über ein Bezugsobjekt (Referenz)
- Klassenmethode: überall innerhalb der Klasse über einfachen Namen oder außerhalb der Klasse über Klassennamen oder Referenz

# Beispiel (Zugriff *innerhalb* der Klasse)

```
public class Unterschied {

    public static int variableKlasse = 1;    // Klassenvariable
    public        int variableInstanz = 2;   // Instanzvariable

    public void methodeInstanz() {           // Instanzmethode
        int blockLokal;                      // blocklokale Variable
        blockLokal = 3;                      // Zugriff auf blockl. Variable
        variableKlasse = 4;                  // Zugriff auf Klassenvariable
        variableInstanz = 5;                 // Zugriff auf Instanzvariable
        methodeKlasse();                     // Zugriff auf Klassenmethode
        methodeInstanz();                    // Zugriff auf Instanzmethode
    }

    public static void methodeKlasse() {    // Klassenmethode
        int blockLokal;                      // blocklokale Variable
        blockLokal = 6;                      // Zugriff auf blockl. Variable
        // hier ist kein Zugriff auf Instanzvariable/-methode moeglich!
        variableKlasse = 7;                  // Zugriff auf Klassenvariable
        methodeKlasse();                     // Zugriff auf Klassenmethode
    }
}
```

# Beispiel (Zugriff *außerhalb* der Klasse)

```
public class Unterschied {
    public static int variableKlasse = 1;    // Klassenvariable
    public        int variableInstanz = 2;   // Instanzvariable
    public void methodeInstanz() {}          // Instanzmethode
    public static void methodeKlasse() {}    // Klassenmethode
}
```

```
class AndereKlasse {
    public static void main(String[] args) {
        // ueber Klassennamen auf Klassenvariable und -methode
        Unterschied.variableKlasse = 8;
        Unterschied.methodeKlasse();

        // ueber Instanz auf Instanz-/Klassenvariablen und -methoden
        Unterschied u = new Unterschied();
        u.variableInstanz = 9;
        u.variableKlasse = 10;
        u.methodeInstanz();
        u.methodeKlasse();
    }
}
```

Später: weitere Einschränkungen durch Vergabe von Zugriffsrechten.

# Zwischenstand

- Durch den Gebrauch des Schlüsselworts static in einer Attribut-/Methodendeklaration erhält man Klassenvariablen/-methoden
- Diese existieren einmal, unabhängig von der Anzahl instanziierter Objekte
- Zugriff außerhalb der eigenen Klasse geschieht über den Klassennamen
- Variablen eines Referenztyps enthalten immer nur Referenzen (inklusive `null`)
- Über eine Referenz und einen Punktoperator hat man Zugriff auf die Instanzvariablen und –methoden der Instanz

**Reflektion**

- Geben Sie eine typische Nutzung für eine Instanz- und eine Klassenvariable an.

# Zusammenfassung

- Klassen sind Beschreibungen gleichartiger Objekte. Sie dienen nur als Hilfsmittel zur einmaligen Beschreibung.
- Ein Objekt ist eine Instanz einer Klasse. Objekte sind die Dinge, mit denen man eigentlich arbeiten will.
- Unterscheidung zwischen Instanz-/Klassenvariablen und Instanz-/Klassenmethoden
- Konstruktoren sind Methoden zur Erzeugung eines Objekts
- Variablen eines Referenztyps (alle Typen außer primitive Typen) enthalten immer einen Zeiger, der auch `null` sein kann

# Sie machen sich klar, was ist / wofür ist … gut

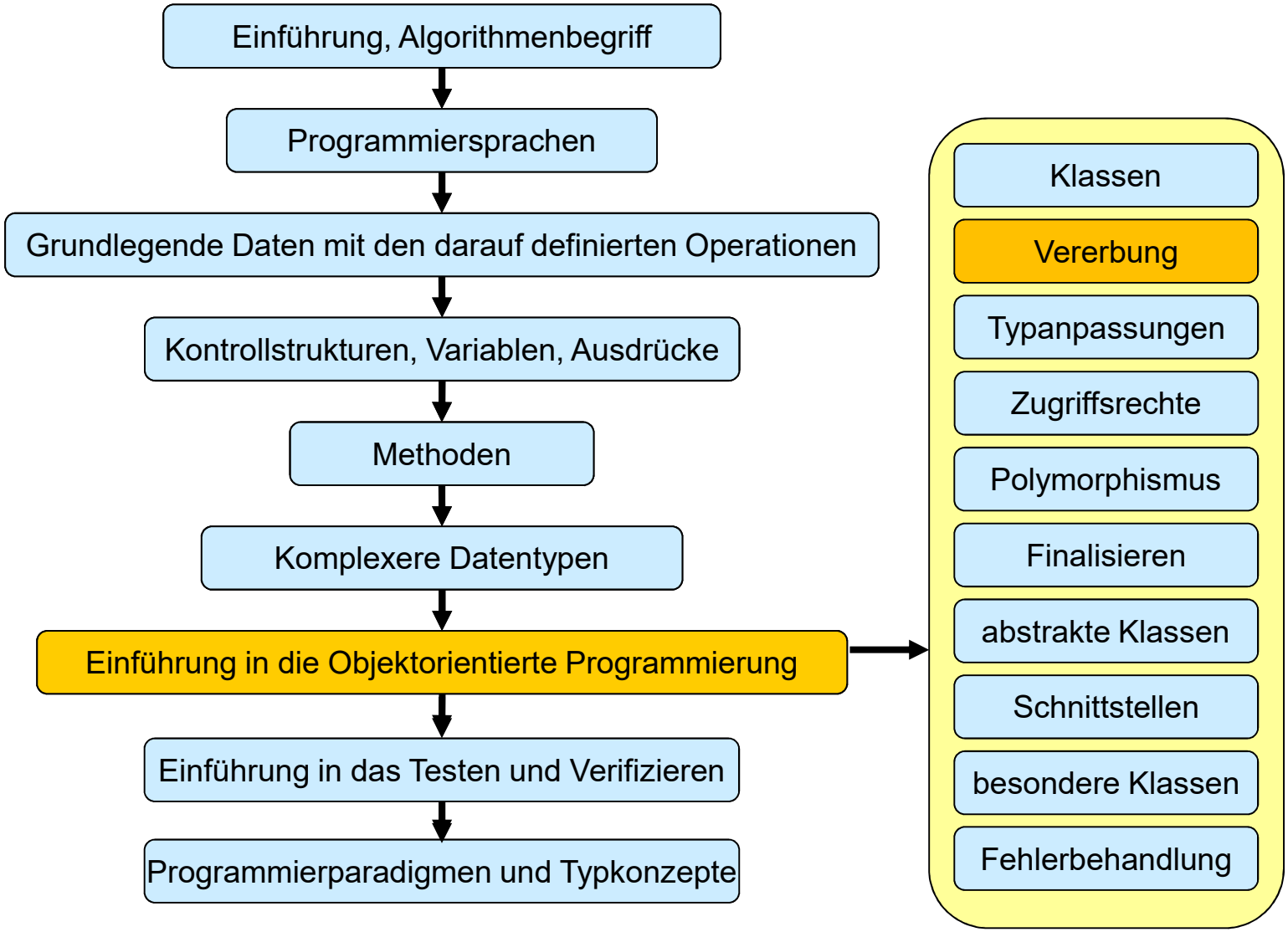
- Klasse, Attribut / Instanzvariable, Instanzmethode, Klassenvariable, Klassenmethode
- Instanz / Objekt, instanziieren, Bezugsobjekt, Referenzvariable
- new-Operator, Konstruktor, Default-Konstruktor
- Lebendauer / Gültigkeit von ... Variablen

- Beantworten Sie beide Fragen der Überschrift für jedes Stichwort in den nächsten Tagen.

- Geben Sie dazu jeweils auch ein konkretes, komplettes, lauffähiges Java-Programm an (was wir so noch nicht hatten), wo genau eines dieser Stichwörter sinnvoll genutzt wird (evtl. in Kombination mit der Umsetzung anderer Wörter).

- Wenn Sie diese Begriffe nicht **jetzt** verstehen und selbstständig anwenden können, werden Sie den Stoff aller folgenden Vorlesungen nicht verstehen!

# Inhalt dieser Veranstaltung



# Vererbung

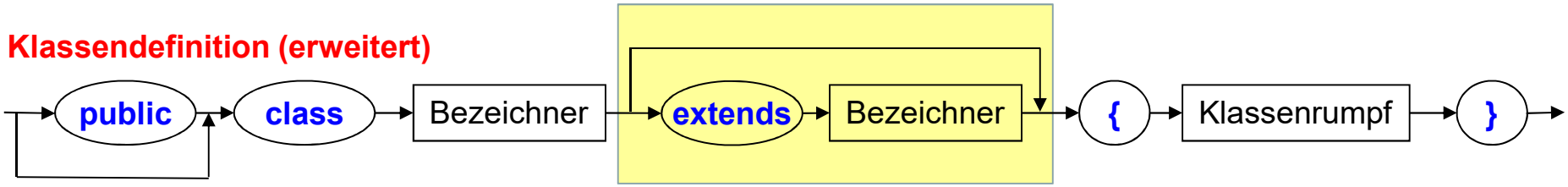
- Im Zoo gibt es Patentiere. Das sind normale Tiere mit allen Eigenschaften und Fähigkeiten, sie haben aber zusätzlich einen Paten und damit ein Einkommen
- Ein Punkt in 3D ist ein 2D Punkt mit einer weiteren Koordinate

- Ansatz 1 (schlecht; wieso?): erzeuge Klasse `Patentier` / `Punkt3D` und kopiere Inhalt aus Klasse `Tier` / `Punkt` dort rüber und füge Erweiterungen / Änderungen ein

- Ansatz 2: gebe einfach an, dass ein Patentier / 3D Punkt ein normales Tier / ein normaler Punkt ist und spezifiziere dann die **zusätzlichen** Eigenschaften und Fähigkeiten

- In der objektorientierten Programmierung dazu: Konzept der Vererbung

# Vererbung in Java



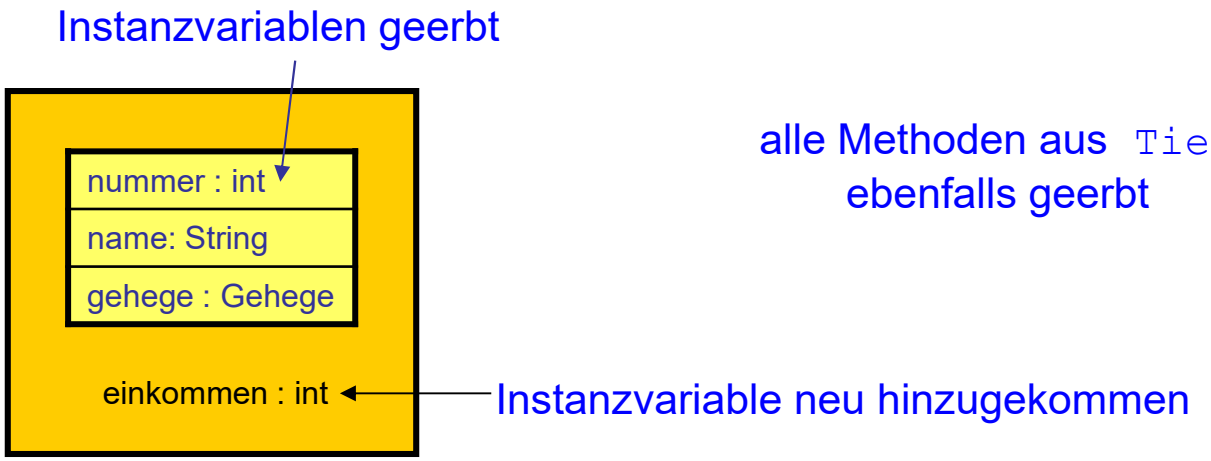

- Eine Klasse $K_1$ kann von genau einer weiteren Klasse $K_2$ abgeleitet werden (erbt von ihr)
- $K_1$ nennt man die abgeleitete Klasse, $K_2$ ist die Oberklasse / Basisklasse von $K_1$
- Mit dieser Kennzeichnung sind alle Variablen und Methoden der Oberklasse auch in der Unterklasse vorhanden und können sogar über ihren einfachen Namen angesprochen werden (erweiterter Gültigkeitsbereich)

- Aber Zugriffsregeln beachten (später mehr; z.B. durch `private` in der Oberklasse sind die Variablen/Methoden nicht sichtbar in der abgeleiteten Klasse)

- Hinweis: in einigen Sprachen wie C++ (aber nicht in Java) ist auch Mehrfachvererbung möglich

# Beispiel (Version 1)

```
public class Patentier extends Tier {

  int einkommen; // jaehrliches Einkommen

  public Patentier( int einkommenPatentier) {
    // parameterloser Konstruktor der Oberklasse wird hier implizit aufgerufen

    einkommen = einkommenPatentier;
  }
}
```



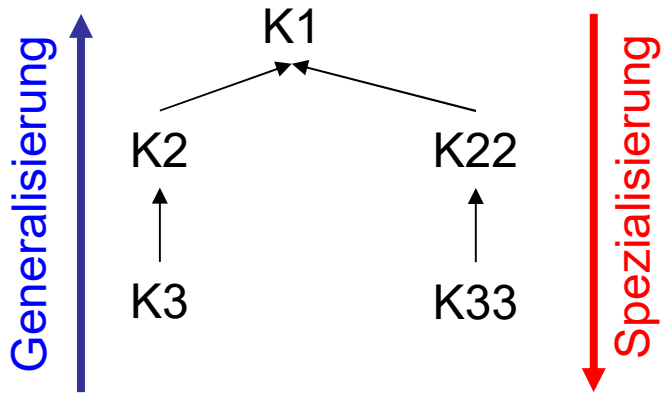

alle Methoden aus `Tier` ebenfalls geerbt

# Klassenhierarchien

- Eine abgeleitete Klasse kann selbst wieder Oberklasse einer weiteren Klasse sein
- Die Erfahrung in der Praxis hat jedoch gezeigt, dass tiefe Klassenhierarchien nur sehr schlecht handhabbar sind
- Grund: Änderungen in der obersten Klasse $K_1$ haben Auswirkungen auf eine Klasse $K_n$ tief unten in der Hierarchie, deren Programmierer noch nie etwas von $K_1$ gehört hat

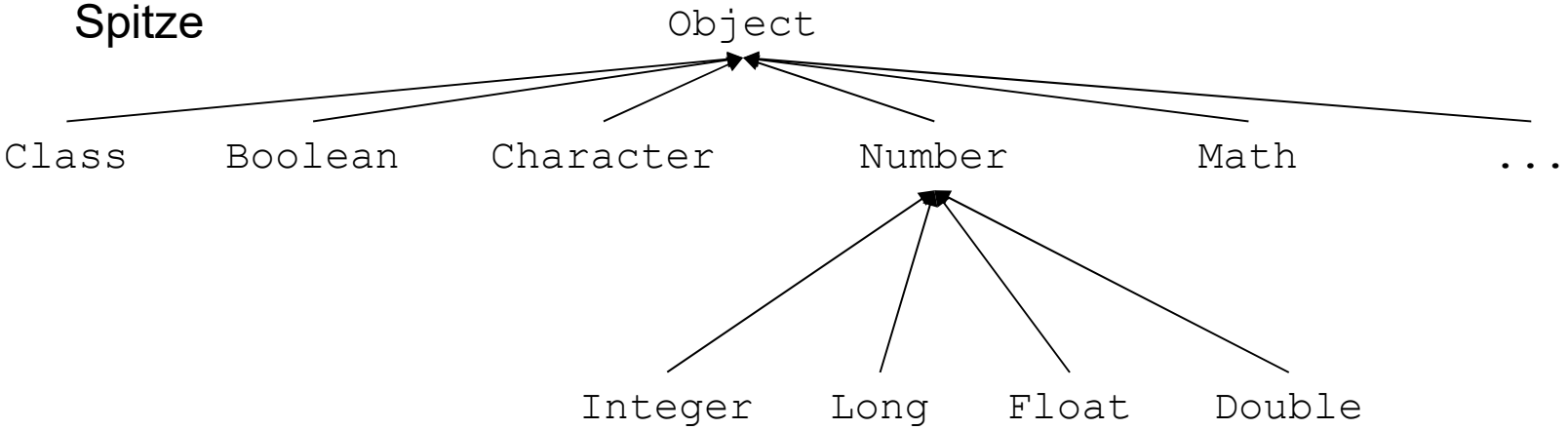
- Beispiel für Hierarchie:

```
public class K1 {   public int x1; }

public class K2 extends K1 { public int x2; }
public class K3 extends K2 { public int x3; }

public class K22 extends K1 { public int x22; }
public class K33 extends K22 { public int x33; }
```



# Klassenhierarchie in Java

- In den Klassen des Java SDK gibt es sehr, sehr viele Klassen, die alle in einer Klassenhierarchie angeordnet sind mit der Klasse `Object` an der Spitze




- Folgerung: alle Methoden und Instanzvariablen der Klasse `Object` sind allen Klassen/Objekten bekannt.

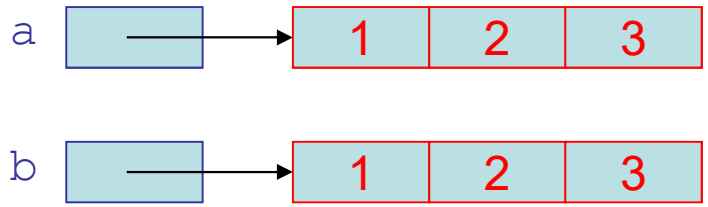
- Hinweis: Andere Programmiersprachen handhaben das anders (keine gemeinsame Wurzel).

# Ausgesuchte Methoden der Klasse `Object`

- Methode `public String toString()`:
  Liefert das Objekt in Form einer Stringdarstellung. Sinnvoll ist das, wenn in einer abgeleiteten Klasse die Methode `toString()` überschrieben wird. Ansonsten liefert die Methode zu Referenzen einen String der Form `java.lang.Object@3be67280`, wobei der Teilstring hinter dem @ die Adresse/Referenz in Hexadezimaldarstellung angibt.
  `toString()` wird z.B. zu einem Objekt `o` aufgerufen in einem Ausdruck der Form `"hallo " + o`

- Methode `protected Object clone()`:
  Klone ein Objekt, z.B. ein Feld identischen Inhalts (erzeuge ein neues Objekt und kopiere Inhalt). Die Klasse des zu clonenden Objektes muss gewisse Kriterien erfüllen (`implements Cloneable`), die für Felder gegeben sind.
  Beispiel:

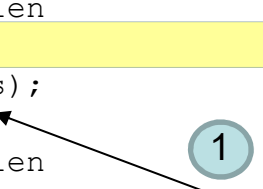
```
int[] a = {1,2,3};
int[] b = a.clone();
```



# this

- Innerhalb einer Klasse kann man explizit das Bezugsobjekt angeben: `this`
- Anwendungen dazu:
  1. Bezugsobjekt angeben, z.B. als Argument einer Instanzmethode
  2. Instanzvariable ansprechen, falls deren Name gültig, aber unsichtbar ist
  3. Konstruktor der eigenen Klasse aufrufen: `this()` oder `this("unbekannt")`
- Beispiel:

```
public class Tier {
    ...
    Gehege gehege;          // Gehege, in dem es wohnt
    ...
    public void umziehen(Gehege gehege) {
        // altem Gehege Auszug mitteilen
        if(this.gehege != null)
            this.gehege.ausziehen(this);

        // neuem Gehege Einzug mitteilen
        if(gehege != null)
            gehege.einziehen(this);

        // aktuelles Gehege merken
        this.gehege = gehege;
    }
}
```



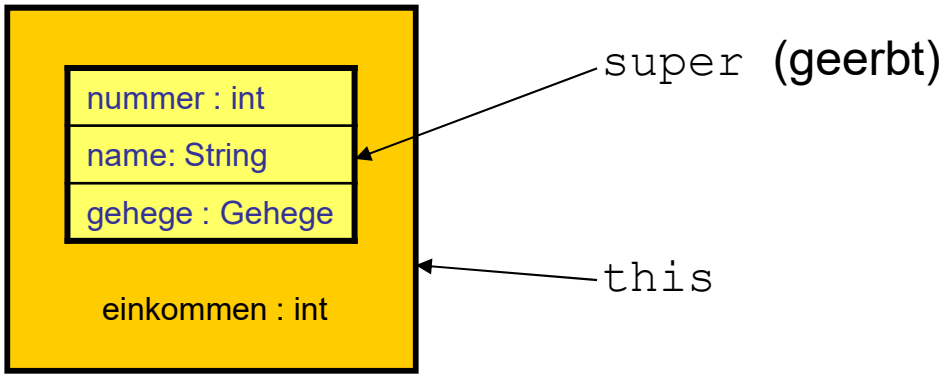

Name `gehege` ist in Methode zweifach gültig, aber es kann immer nur ein Name sichtbar sein (hier: Parameter der Methode)

Objekt (bzw. Referenz darauf) wird übergeben

# super

- Man kann explizit auf den Teil eines Objekts Bezug nehmen, der durch die Oberklasse abgedeckt wird: `super`
- `super` wird analog zu `this` selbst wie ein Objekt gesehen
- Beispiel für Anwendung (siehe nächstes Beispiel)
  1. Konstruktor der Oberklasse aufrufen (nur als erste Anweisung in einem Konstruktor möglich)
  2. Bei überschriebenen Methodennamen kann man explizit eine bestimmte Methode der Oberklasse damit ansprechen

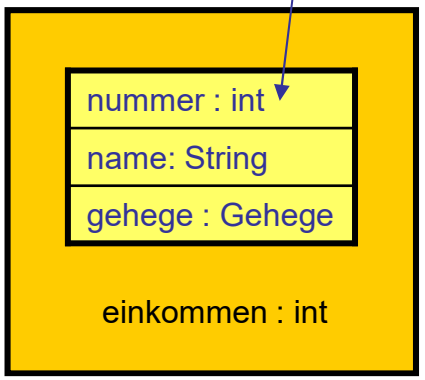
- Beispiel Tierobjekt:



# Beispiel Patentier (Version 2)

```
public class Patentier extends Tier {

    int einkommen;    // jaehrliches Einkommen

    public Patentier( int einkommen) {
        // implizit wird hier aufgerufen: super();
        this.einkommen = einkommen;
    }

    public Patentier(String name, int einkommen) {
1       super(name);  // Konstruktor aus Oberklasse
        this.einkommen = einkommen;
    }

    public String toString() {
2       return super.toString() // Methode der Oberklasse
                + ", Einkommen=" + einkommen;
    }
}
```

Instanzvariablen geerbt

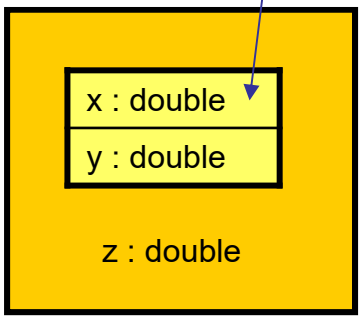



alle Methoden aus `Tier` ebenfalls geerbt

# Beispiel 3D Punkt

```
public class Punkt3D extends Punkt {

    // zusaetzliche Koordinate des Punktes
    private double z;

    // Konstruktor
    public Punkt3D(double x, double y, double z) {
        super(x,y);  // Konstruktor der Oberklasse
        this.z = z;
    }

    // liefere z-Koordinate
    public double getZ() {
        return z;
    }

    // verschiebe den Punkt
    public void verschieben(double dx, double dy, double dz) {
        super.verschieben(dx, dy); // Methode der Oberklasse
        z += dz;
    }

    // liefere eine Textdarstellung des Punktes
    public String toString() {
        return "(" + getX() + "," + getY() + "," + getZ() + ")";
    }
}
```

geerbt
(aber wegen `private` in `Punkt` nicht sichtbar)




alle Methoden aus `Punkt` ebenfalls geerbt

# Zwischenstand

- Vererbung dient der Ausgliederung von Teilbeschreibungen
- In einer abgeleiteten Klasse werden zusätzliche Eigenschaften / Fähigkeiten beschrieben
- Wird in einem Konstruktor einer abgeleiteten Klasse nicht an erster Stelle im Code explizit ein Konstruktor der Oberklasse aufgerufen, so wird implizit der parameterlose Konstruktor der Oberklasse aufgerufen
- In Java gibt es eine Klassenhierarchie mit der Klasse `Object` an der Spitze

**Reflektion**

- Welche Vor- und Nachteile hat es, dass es in Java eine „Mutter aller Klassen“ gibt?