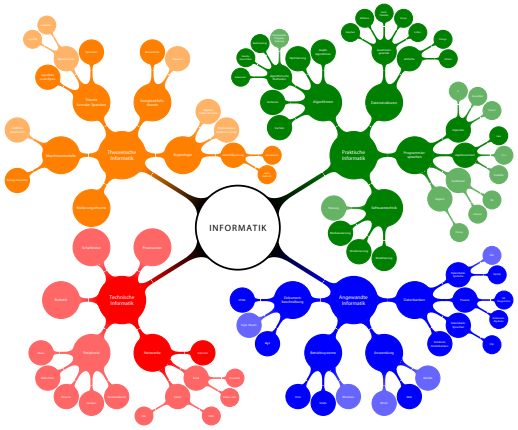
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 22

Objekte – Attribute und Lebenszyklus

Ihre Klassen, ihre Attribute, ihr Lebenszyklus

Vorlesung Einführung in die Informatik 1 vom 21. Januar 2014 von Till Tantau



Lernziele von Kapitel 22

1. Modellierung von Daten mittels Klassen verstehen
2. Konzept des Objekts und des Attributs kennen und anwenden können
3. Lebenszyklus von Objekten verstehen

Gliederung von Kapitel 22

Alles ist ein Objekt.

- In der Wirklichkeit begegnen uns ständig *Dinge* wie Tische, Stühle, T-Zellen, Antigene und so weiter.
- Die Idee der *objektorientierten Programmierung* ist, solche Dinge in der Programmiersprache *möglichst direkt zu repräsentieren*.
- Dazu benutzt man dann *Objekte*.

Beispiel

Wollen wir zwanzig Antigene und drei Antikörper simulieren, so würde das in einem OO-Programm mittels zwanzig Antigen-Objekten und drei Antikörper-Objekten geschehen.

Die zentralen Eigenschaften von Objekten.

1. Objekte haben eine *Identität*.
   - Das bedeutet, dass es mehrere Objekte geben kann mit denselben Eigenschaften.
     Beispiel: Es viele identisch aussehende Norrebo-Regale bei Ikea. Trotzdem ist Ihr Norrebo-Regal-Objekt ein anderes als mein Norrebo-Regal-Objekt. Ihr und mein Norrebo-Regal-Objekt haben eine *unterschiedliche Identität*.
2. Objekte haben *Attribute*.
   - Dies sind Eigenschaften wie *Höhe*, *Breite*, *Oberflächenstruktur* oder was auch immer.
   - Man kann sie *erfragen* und auch *verändern*.
3. Objekte haben *Fähigkeiten*:
   - Man kann Objekten *Nachrichten schicken*.
   - Dazu werden *Objekt-Methoden* benutzt.

   Dies besprechen wir im nächsten Kapitel.

Klassen von Objekten.

- Wir benutzen Klassen bereits zur *Modularisierung*.
- Ihre *eigentliche Aufgabe* ist aber, *Objekte zu Klassen zusammenzufassen*.
- *Dies ist eine ganz andere Aufgabe!*

Idee der Klassenbildung

- Betrachten wir ein Modell des Warenlagers von Ikea.
- Jedes kaufbare Ikea-Ding ist ein Objekt im Modell.
- Man bildet nun *Klassen von gleichartigen Objekten*. Beispiel: Alle Tische bilden eine `Tisch`-Klasse, alle Stühle bilden eine `Stuhl`-Klasse und so weiter.
- Jedes Objekt gehört genau einer Klasse an und *die Klasse legt die Attribute des Objektes fest*.

Zur Diskussion

Wir wollen eine *Antigen-Klasse* bauen. Die Objekte dieser Klasse sollen also Antigene modellieren.
Welche Attribute erscheinen Ihnen sinnvoll/nötig?

Eine einfache Studenten-Klasse.

Nehmen wir an, Ihre Tutorin möchte eine Liste der Studenten in ihrem Tutorium anlegen. Dies ist ein Array von »Studentenobjekten«. Sinnvolle Attribute sind

- Vor- und Nachname und
- Matrikelnummer.

In Java sieht das so aus:

```
class Student
{
  String nachname;        // Erstes Attribut
  String vorname;         // Zweites Attribut
  int    matrikelnummer; // Drittes Attribut

  // Attribute werden wie Variablen deklariert,
  // sind aber keine echten Variablen.

  // Life is hard.
}
```

Klassen sind Datentypen.

Wiederholung: Was ist ein Datentyp?

- Ein *Datentyp* war eine Menge von möglichen Werten.
- Hat ein Wert den Datentyp `int`, so weiß man, dass dieser Wert eine ganze Zahl ist.
- Es gibt nur sehr wenige *eingebaute* Datentypen.
  Beispiel: `int`

Erstellung neuer Datentypen

- Jede Klasse definiert einen *neuen* Datentyp.
  Beispiel: `Student`
- Hat ein Wert den Datentyp `Student`, so weiß man, dass dieser Wert ein Studenten-Objekt ist.
- Man nennt die Objekte auch *Instanzen* ihrer Klasse.

5-Minuten-Aufgabe

Zur Übung

Schreiben Sie den Code einer Klasse `Date` auf.

Der Lebenszyklus eines Objektes

1. Objekte werden irgendwann *erzeugt*.
2. Objekte werden danach *benutzt*.
3. Objekte werden *automatisch vernichtet*, wenn sie nicht mehr benutzt werden.

Die Erzeugung von Objekten mittels `new`.

Irgendwo im Programm:

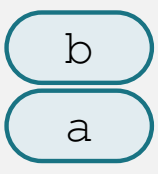
```
Student a, b;
// 1. Deklaration von zwei Variablen (noch keine Objekte!)

a = new Student ();
// 2. Jetzt erzeugen wir ein Objekt

b = a;
// 3. Jetzt verweisen sowohl a als auch b auf das Objekt

b = new Student ();
// 4. Jetzt erzeugen wir ein zweites Objekt
```

1



Die Erzeugung von Objekten mittels `new`.

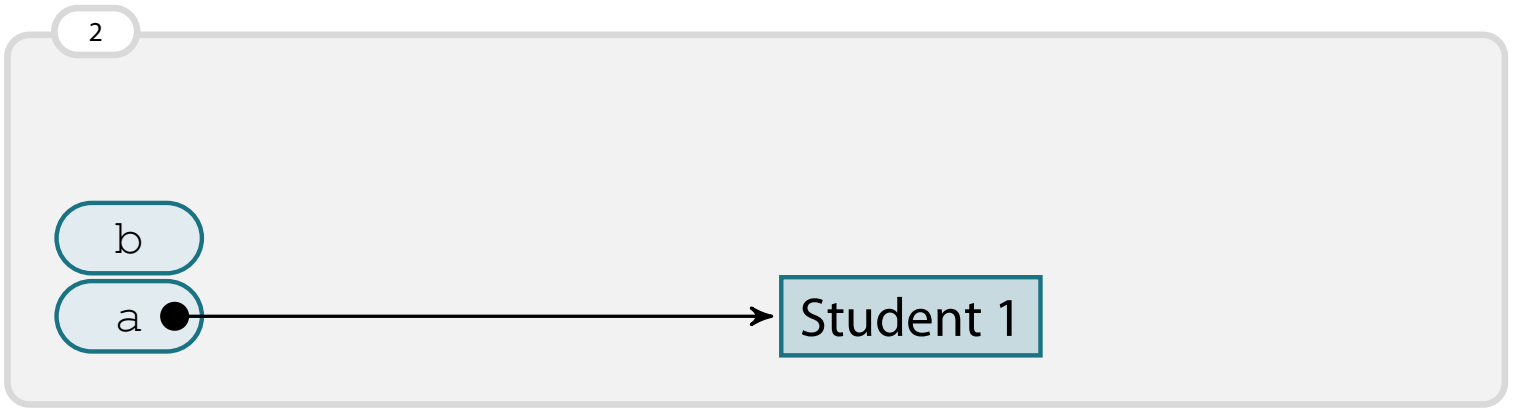
Irgendwo im Programm:

```
Student a, b;
// 1. Deklaration von zwei Variablen (noch keine Objekte!)

a = new Student ();
// 2. Jetzt erzeugen wir ein Objekt

b = a;
// 3. Jetzt verweisen sowohl a als auch b auf das Objekt

b = new Student ();
// 4. Jetzt erzeugen wir ein zweites Objekt
```



Die Erzeugung von Objekten mittels `new`.

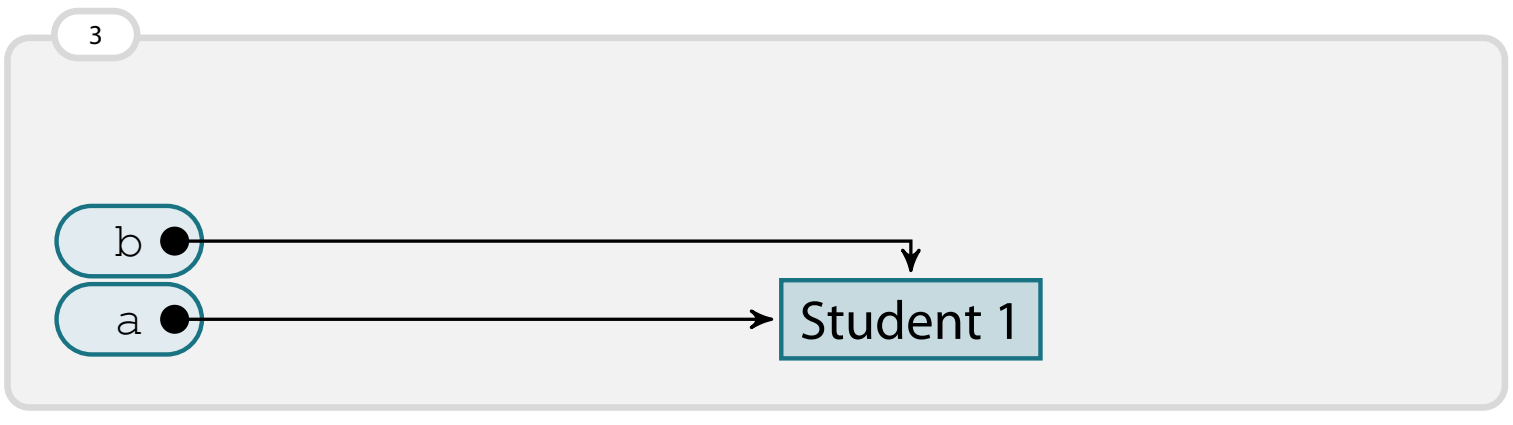
Irgendwo im Programm:

```
Student a, b;
// 1. Deklaration von zwei Variablen (noch keine Objekte!)

a = new Student ();
// 2. Jetzt erzeugen wir ein Objekt

b = a;
// 3. Jetzt verweisen sowohl a als auch b auf das Objekt

b = new Student ();
// 4. Jetzt erzeugen wir ein zweites Objekt
```



Die Erzeugung von Objekten mittels `new`.

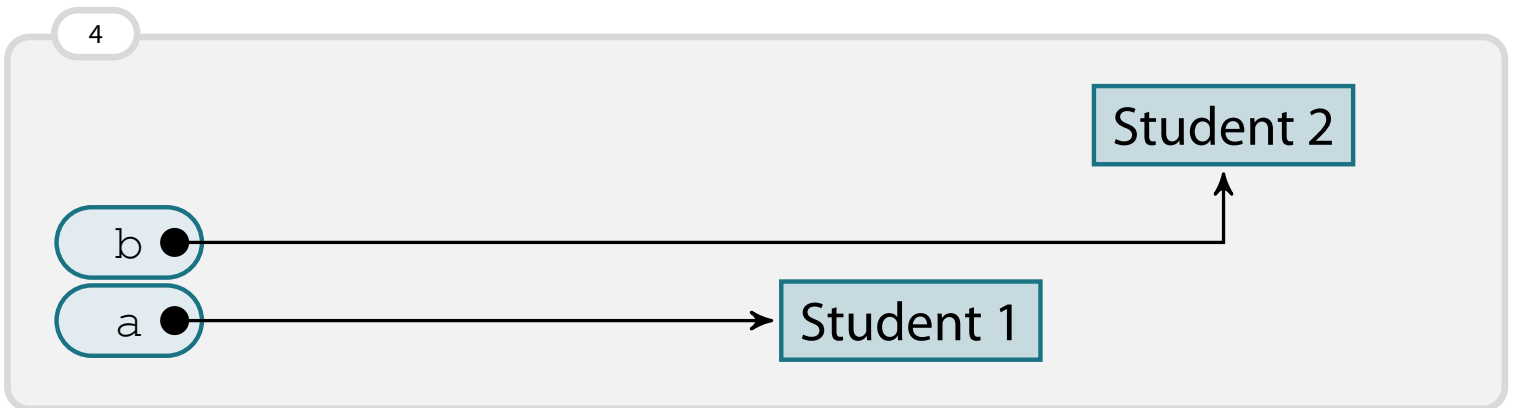
Irgendwo im Programm:

```
Student a, b;
// 1. Deklaration von zwei Variablen (noch keine Objekte!)

a = new Student ();
// 2. Jetzt erzeugen wir ein Objekt

b = a;
// 3. Jetzt verweisen sowohl a als auch b auf das Objekt

b = new Student ();
// 4. Jetzt erzeugen wir ein zweites Objekt
```



Die großen Unterschiede zwischen Variablen und Objekte

Objekte

- Ein *Objekt* wird einmal mittels `new` erzeugt und bleibt dann an einer festen Stelle im Speicher. Lediglich seine Attribute ändern sich.
- Objekte können *sehr lange leben*. Insbesondere *leben sie auch weiter*, wenn die Methode, in der sie erzeugt wurden, beendet ist.

Variablen

- *Variablen* enthalten lediglich *Verweise* auf Objekte, dabei
- können *viele Variablen* auf *dasselbe Objekt* verweisen.
- Variablen können auch *zuerst auf das eine, später auf ein anderes Objekt verweisen*.
- Schließlich können Variablen auch *auf gar kein Objekt* verweisen. (Dann ist diese Variable gleich `null`.)
- Im Gegensatz zu Objekten sind Variablen lokal zu ihrem Scope.

Zugriff auf die Attribute eines Objektes.

- Jedes Objekt speichert für jedes seiner Attribute einen Wert.
- Um diese Attribute zu lesen oder zu schreiben, muss man das Objekt »aufmachen«, genau wie bei Scopes:

Beispiel

```
// Deklaration einer Variable und Erzeugung eines Objekts
Student a = new Student ();

// Setzen des Namens
a.nachname = "Musterfrau";
a.vorname  = "Petra";

// Setzen der Matrikelnummer
a.matrikelnummer = 1234567;

// Ausgeben der Matrikelnummer
System.out.println ("Die␣Matrikelnummer␣ist␣" +
    a.matrikelnummer);
```

Beispiel einer Objektveränderung.


```
// Datei Student.java
class Student
{
  String nachname, vorname;
  int    matrikelnummer;
}

// Datei Beispiel.java
class Beispiel {
  static void exmatrikuliere (Student s)
  {
    s.matrikelnummer = -1;
  }
}
```

5-Minuten-Aufgabe

Zur Übung

Geben Sie den Code einer Methode `konflikt` an, die überprüft, ob zwei Studenten(objekte) die gleiche Matrikelnummer haben, aber unterschiedliche Namen.

```
static boolean konflikt (Student a, Student b)
```

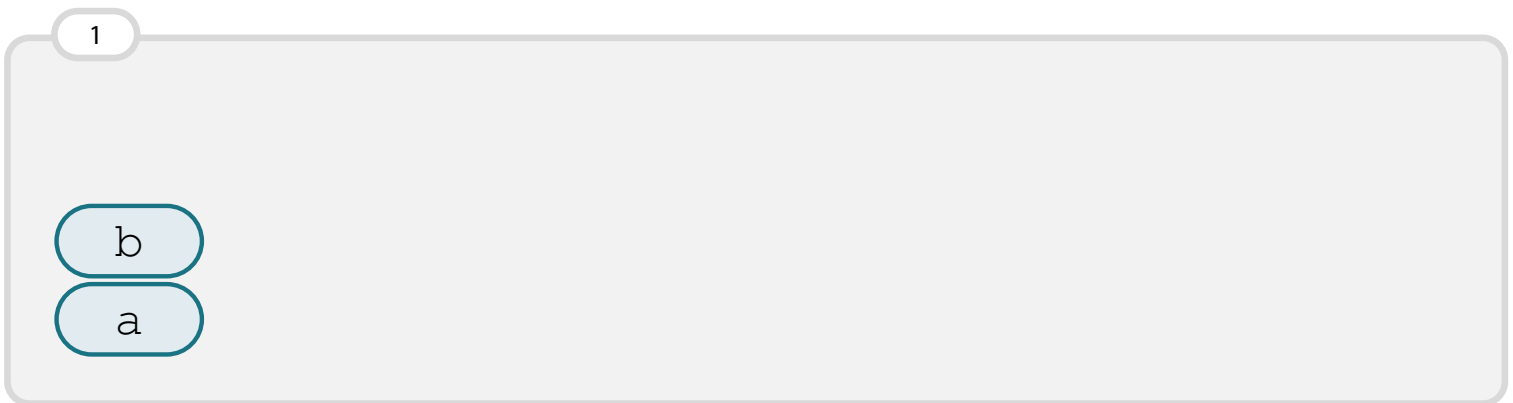
Objekte werden automatisch gelöscht.


```
Student a, b;
// 1. Deklaration von zwei Variablen (noch keine Objekte!)

a = new Student ();
// 2. Jetzt erzeugen wir ein Objekt

b = new Student ();
// 3. Jetzt erzeugen wir ein zweites Objekt

a = b;
// 4. Jetzt kann das erste gelöscht werden
```



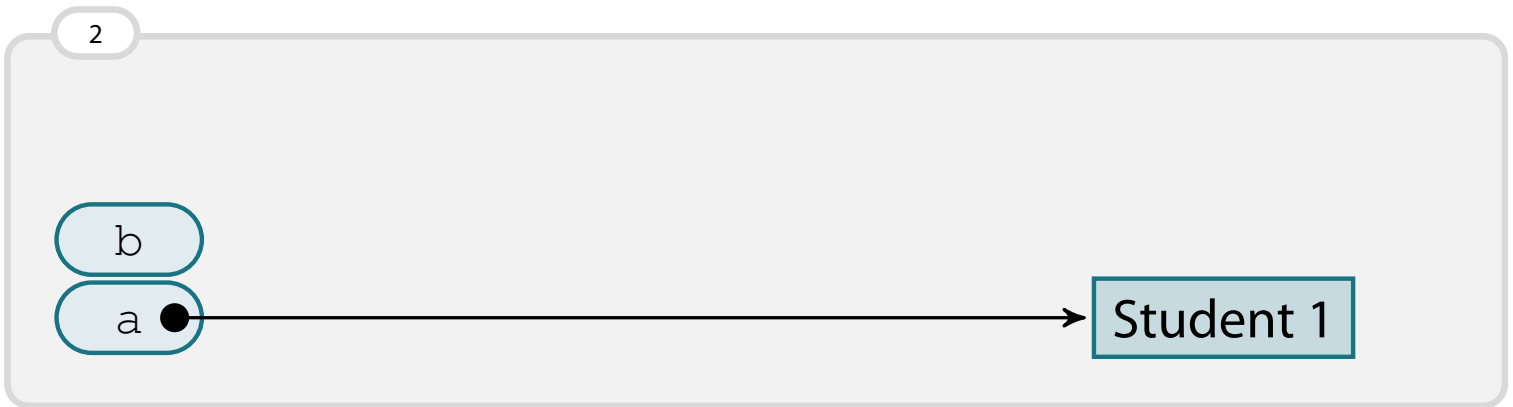
Objekte werden automatisch gelöscht.


```
Student a, b;
// 1. Deklaration von zwei Variablen (noch keine Objekte!)

a = new Student ();
// 2. Jetzt erzeugen wir ein Objekt

b = new Student ();
// 3. Jetzt erzeugen wir ein zweites Objekt

a = b;
// 4. Jetzt kann das erste gelöscht werden
```



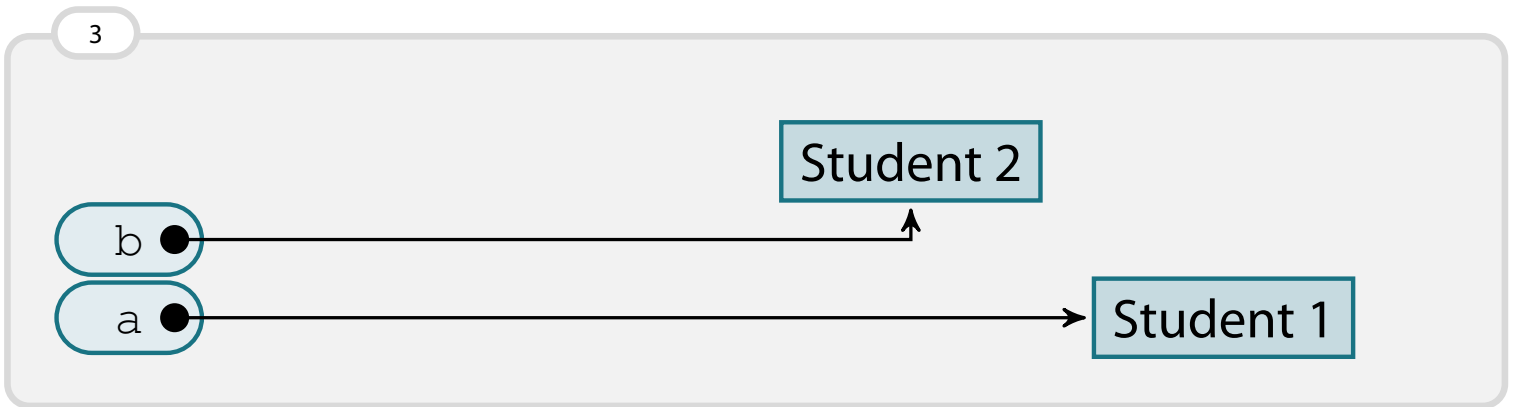
Objekte werden automatisch gelöscht.


```
Student a, b;
// 1. Deklaration von zwei Variablen (noch keine Objekte!)

a = new Student ();
// 2. Jetzt erzeugen wir ein Objekt

b = new Student ();
// 3. Jetzt erzeugen wir ein zweites Objekt

a = b;
// 4. Jetzt kann das erste gelöscht werden
```



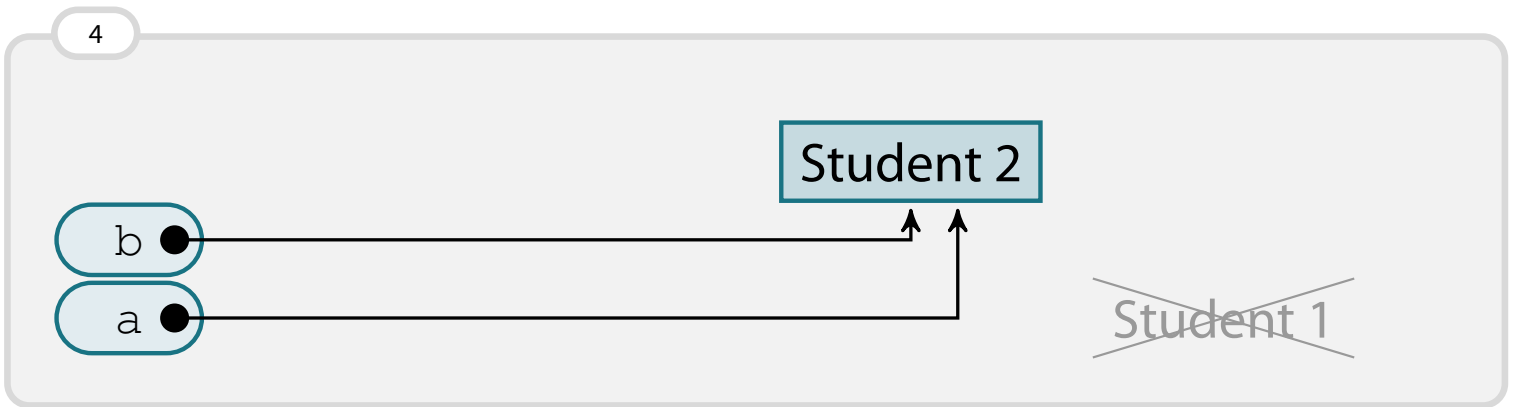
Objekte werden automatisch gelöscht.


```
Student a, b;
// 1. Deklaration von zwei Variablen (noch keine Objekte!)

a = new Student ();
// 2. Jetzt erzeugen wir ein Objekt

b = new Student ();
// 3. Jetzt erzeugen wir ein zweites Objekt

a = b;
// 4. Jetzt kann das erste gelöscht werden
```



Objekte können Objekte als Attribute haben.

- Manche Objekte »bestehen« aus anderen Objekten. (Man sagt auch, sie sind *aggregiert* aus anderen Objekten.)
- Zum Beispiel besteht eine Zelle aus verschiedenen Objekten: Dem Zellkern, der Membran, den Mitochondrien, etc.
- Dieser Umstand lässt sich auch im Programm abbilden:
  - Es gibt `Cell`-Objekte, `Nucleus`-Objekte, `Membrane`-Objekte und `Mito`-Objekte.
  - Ein `Cell`-Objekt muss sich nun merken, aus *welchen* der anderen Objekte es »besteht«.
  - Dazu erhält das `Cell`-Objekte *Attribute*, die auf die anderen Objekte verweisen.

Code der Kern-, Membran- und Mitochondrienklassen.


```
class Nucleus {
  String dna;
  double weight;
  // ...
}

class Membrane {
  String kind;
  // ...
}

class Mito {
  String dna;
  double weight;
  double energy_level;
  // ...
}
```

Erweiterter Code der Zellklasse.

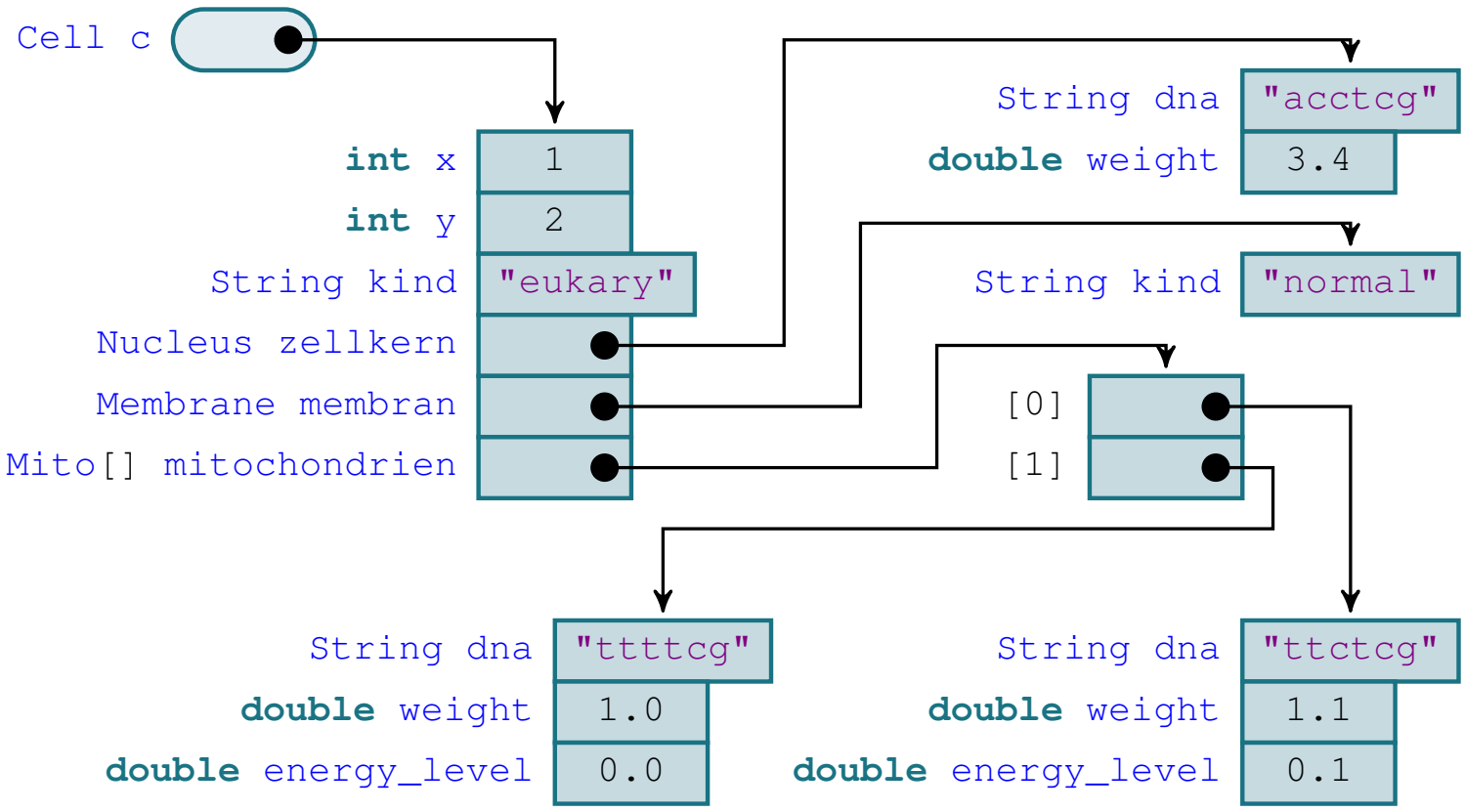

```
// Datei Cell.java
class Cell {
  int x, y;
  String kind;

  // Verweis auf das Nucleus Objekt:
  Nucleus zellkern;

  // Verweis auf die Membran:
  Membrane membran;

  // Array von Verweisen auf die Mitochondrien
  Mito[] mitochondrien;
}
```

Visualisierung der Objekthierarchie eines Zellobjekts.



Code zur Erzeugung der Objekthierarchie.


```
Cell c     = new Cell ();     // Zellobjekt erzeugt, aber noch leer
c.x        = 1;
c.y        = 2;
c.kind     = "eukary";

Nucleus n = new Nucleus (); // Jetzt das Zellkernobjekt
n.dna      = "acctcg";
n.weight   = 3.4;

c.zellkern = n; // Jetzt verweist c.zellkern auf das n Objekt

c.membran = new Membran (); // Erzeugung und Zuweisung in einem
c.membran.kind = "normal";  // Die ''kind'' der Membran von c

c.mitochondrien = new Mito[2]; // Erzeuge das Array-Objekt

c.mitochondrien[0] = new Mito (); // Erzeuge erstes Mito-Objekt
c.mitochondrien[0].dna = "ttttcg";
c.mitochondrien[0].weight = ...;

c.mitochondrien[1] = new Mito (); // Erzeuge zweites Mito-Objekt
c.mitochondrien[1].dna = "ttctcg";
c.mitochondrien[1].weight = ...;
```

Ein Knopf hält Verweise auf Farbobjekte.


```
class Color
{
  float red_part;   // Rot-Anteil; Zahl zwischen 0 und 1
  float green_part; // Grün-Anteil
  float blue_part;  // Blau-Anteil
}

class Button
{
  int height;
  int width;

  Color background; // Verweis auf ein Objekt.
  Color text_color; // Noch ein Verweis auf ein Objekt
                    // (vielleicht sogar auf dasselbe,
                    // aber besser nicht...)

  String text;      // Auch Strings sind in Wirklichkeit
                    // Objekte
}
```

5-Minuten-Aufgabe

Zur Übung

Geben Sie Code an, um `Button`- und `Color`-Objekte zu erzeugen, so dass das `Button`-Objekt korrekte Verweise die Vorder- und Hintergrundfarbe hat.

Zusammenfassung

Klassen sind Datentypen

Klasse sind *Datentypen* (so wie `int`), deren »Werte« (so wie `5` bei `int`) man *Objekte* nennt.

Klassen – und damit auch Objekte – haben *Attribute*

In der Klasse deklariert man Attribute genau so, wie man Variablen deklariert.

```
class Example {
  int    attribute_1;
  String attribute_2;
}
```

Der Lebenszyklus eines Objekts

```
// Geburt
Example my_object = new Example ();

// Leben: Schreib-Zugriff
my_object.attribute_1 = 5;
my_object.attribute_2 = "Hallo";

// Leben: Lese-Zugriff
if (my_object.attribute_1 > 0) {
   ...
}

// Tod: automatisch
```