Begriffe und Beispiele

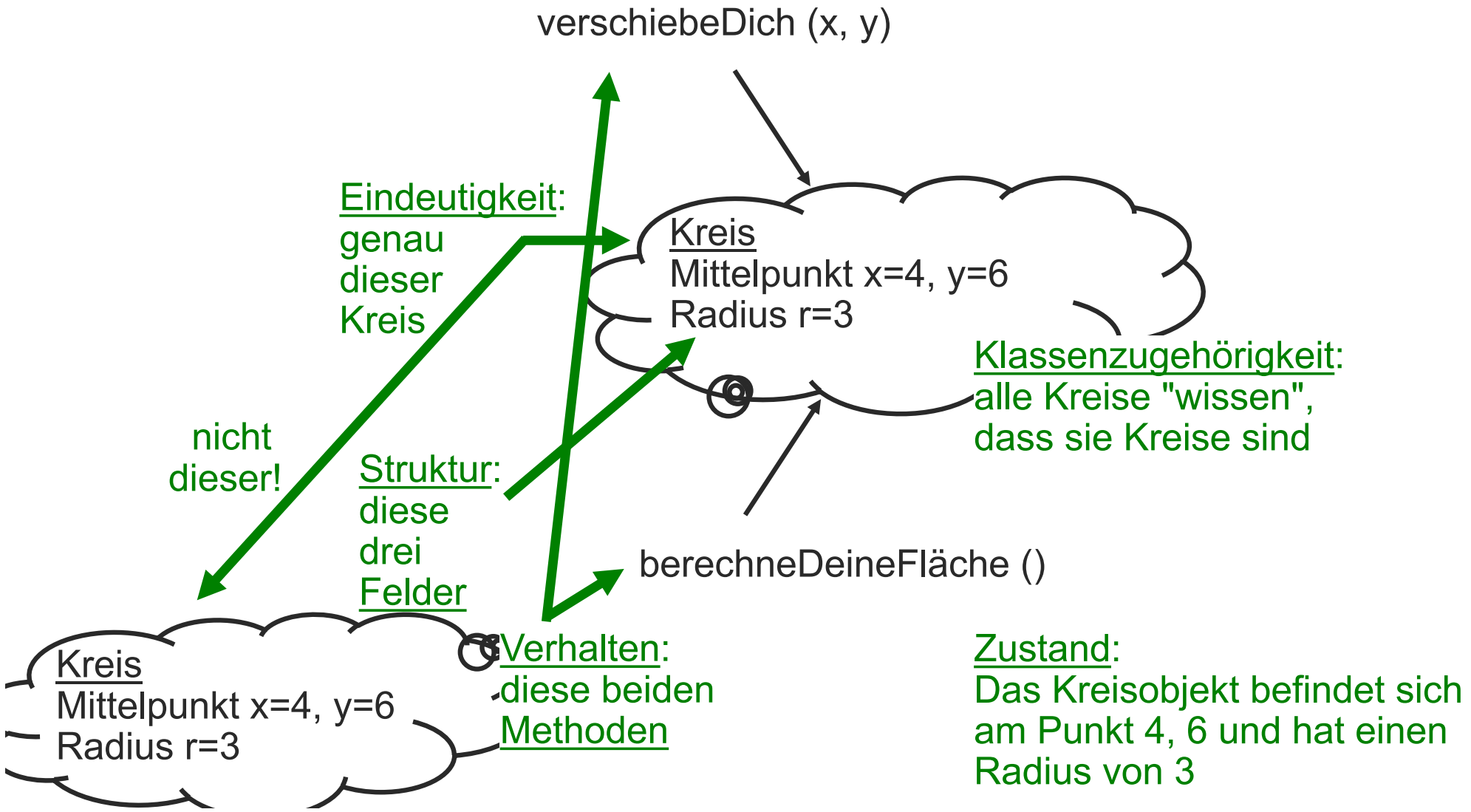
- Objekte
- Klassen
- Konstruktoren…
  - …ohne Parameter
  - …mit Parametern
  - Default- und Initialwerte für Felder
  - Bemerkungen zu Konstruktoren
- Ein Beispiel
  - Benutzerdefinierter Konstruktor
  - Das Schlüsselwort *this* und seine Bedeutung

Definition: Objekt

Ein ***Objekt***

- ist *eindeutig* identifizierbar,
- gehört zu einer *Klasse*,
- implementiert […] *Methoden* (→ sein *Verhalten*) und
- hat einen änderbaren *Zustand* in Form von Werten seiner *Felder* (→ seine *Struktur*).

Die Begriffe am Beispiel



Der Programmcode dazu

```
class Kreis {

    private int x, y, r;

    Kreis ( int xKoordinate, int yKoordinate, int radius ) {
        x = xKoordinate;
        y = yKoordinate;
        r = radius;
    }

    double berechneDeineFläche () {
        return r * r * 3.14;
    }

    void verschiebeDich ( int dX, int dY ) {
        x += dX;
        y += dY;
    }

}
```

Definition: Klasse

Eine ***Klasse*** definiert die (Daten-) *Struktur* und das *Verhalten* aller

- aus ihr instanziierten /
- von ihr abgeleiteten /
- nach ihrem Bauplan erstellten

Objekte

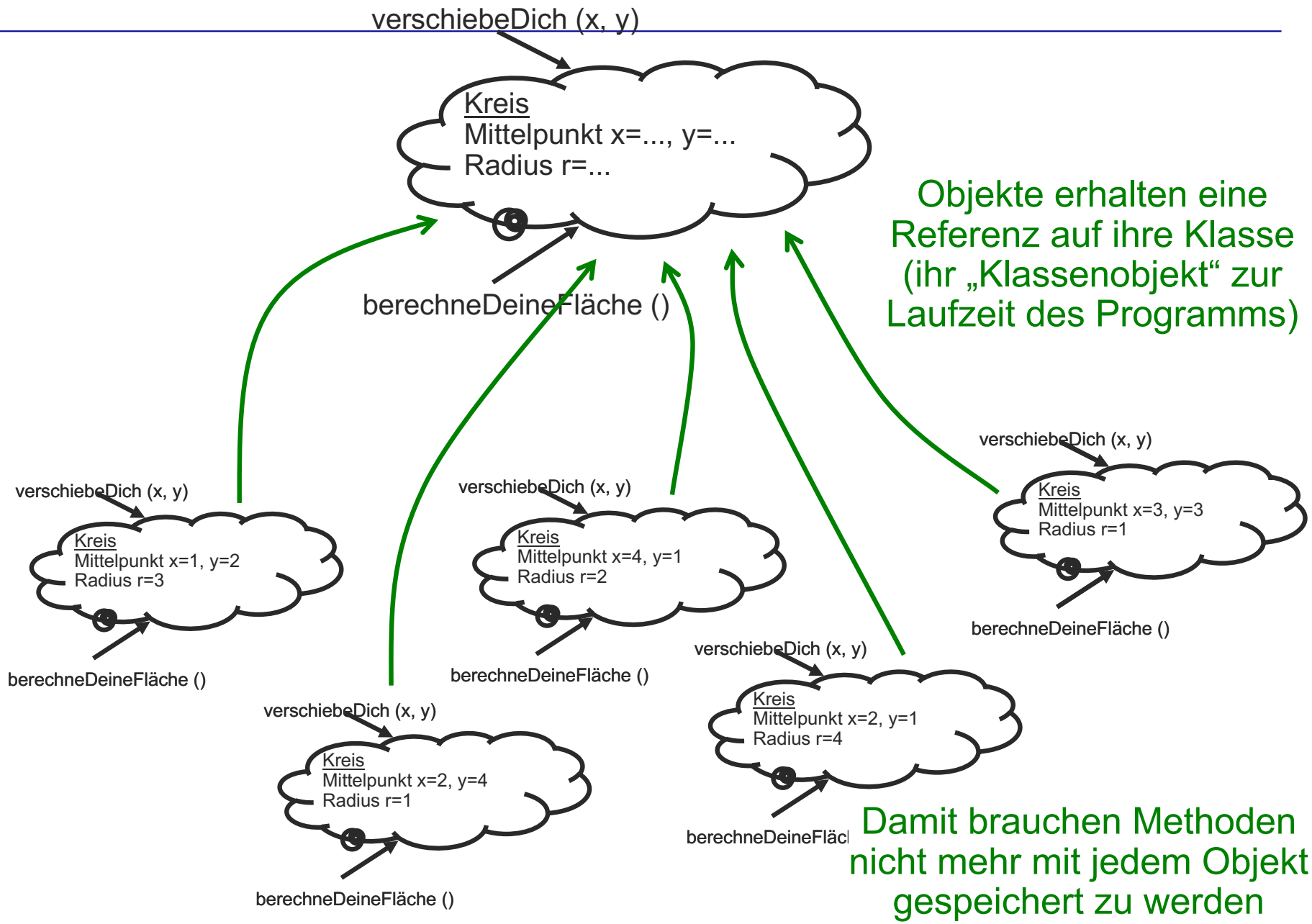
Verwendung von Klassen

- Zur Erinnerung aus der Definition von *Objekt*:
  Ein Objekt
  - ... gehört zu einer *Klasse*, ...

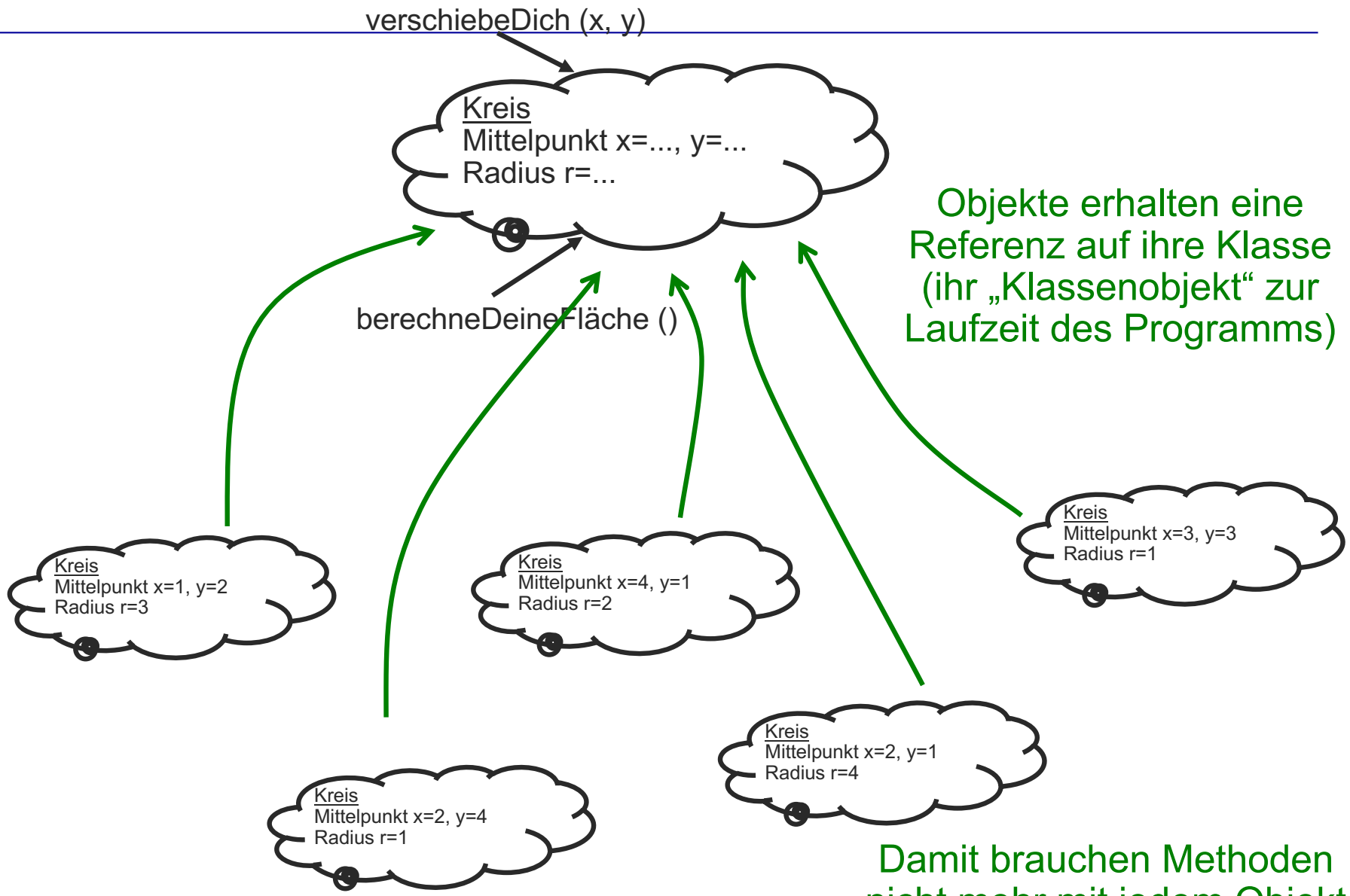
- Beim *Entwickeln* eines Programms:
  Objekte mit der gleichen Struktur (= den gleichen Feldern)
  und dem gleichen Verhalten (= implementierte Methoden)
  werden durch eine Klasse beschrieben

- Zur *Laufzeit* eines Programms:
  Gleichartige Objekte werden erzeugt, indem sie aus der
  gleichen Klasse instanziiert werden

Beispiel für Klassen und Objekte



Beispiel für Klassen und Objekte



Objekte erhalten eine Referenz auf ihre Klasse (ihr „Klassenobjekt“ zur Laufzeit des Programms)

Damit brauchen Methoden nicht mehr mit jedem Objekt gespeichert zu werden

Klassen und Objekte 1/2

- Klassen-Deklarationen sind *benutzerdefinierte* (Daten-) *Typ-Vereinbarungen*
- *Benutzerdefinierte* Datentypen sind zu verstehen als Ergänzung zu den *vordefinierten* Datentypen wie z.B. int, boolean, String
- Benutzerdefinierte Datentypen können, sobald sie einmal deklariert sind, wie vordefinierte Datentypen verwendet werden, um zum Beispiel Variablen dieses Typs zu deklarieren

```
CD meineCd;
```

- Die Werte, die eine Klasse annehmen kann, bezeichnet man als *Objekte* oder *Instanzen*: wir sprechen von einem Buch-Objekt, einem Auto-Objekt, einem Adress-Objekt etc.
- Daher spricht man von der objektorientierten Programmierung (OOP) und von *objekt*orientierten Programmiersprachen (wie zum Beispiel Java, C++, C#)

Klassen und Objekte 2/2

- Objekte, also Werte von Klassen-Typen, werden, genau wie Arrays, durch einen *Konstruktor*-Aufruf mit dem Schlüsselwort *new* erzeugt (man sagt auch kreiert, abgeleitet, instanziiert)

```
meineCd = new CD();
```

Die runden Klammern sind Parameter-Klammern für die aktuellen Parameter (wie bei Methodenaufrufen)

Konstruktoren

- Objekte, also Werte von Klassen-Typen, werden genau wie Arrays durch einen *Konstruktor*-Aufruf mit dem Schlüsselwort *new* erzeugt (oder kreiert, abgeleitet, instanziiert)

```
meineCd = new CD();
```

  Beim Konstruktor-Aufruf können aktuelle Parameter in runden Klammern übergeben werden

- Korrekt gesagt, besteht der Konstruktor-Ausdruck aus
  - dem Schlüsselwort *new* mit
  - dem *Typnamen*, also dem Namen der zu instanziierenden Klasse (hier *CD*), mit
  - eventuellen *Parametern* in runden Klammern (hier keine)

Benutzerdefinierte Konstruktoren

- Ein Entwickler kann *eigene Konstruktoren* definieren
- Diese *können* beliebige Programmaktionen ausführen, um Werte für die Initialisierung der Felder der Klasse zu bestimmen:
  - Dateien öffnen und Konfigurationsdaten auslesen
  - beim Benutzer nachfragen
  - das aktuelle Datum/die aktuelle Uhrzeit benutzen…
  - etc.

- Benutzerdefinierte Konstruktoren können *beim Aufruf mit aktuellen Parametern versorgt* werden, welche für die Initialisierung der Felder der Klasse genutzt werden (genauso wie Methoden Parameter erhalten können)

Beispiel für eine Konstruktor-Definition ohne Parameter

```
class Book {
    private String    author = "";
    private String    title = "";
    private int       yearOfPublication;
    private boolean   present = true;
    private Date      returnDate;

    Book ()
    {
            author = "unknown";
            yearOfPublication = 1998;
            returnDate = new Date();
    }

}
```

- Die Werte der Felder nach der Konstruktion eines Objekts der Klasse Book
  - author "unknown"
  - title ""
  - yearOfPublication 1998
  - present true
  - returnDate Objekt der Klasse *Date*

Was passiert mit Feldern, die nicht initialisiert werden?

Wir erinnern uns: nicht initialisierte *Variable*

- Ist eine *lokale Variable* in einer Methode *nicht initialisiert*, dann meldet der Compiler einen *Fehler* beim Versuch, lesend darauf zuzugreifen

  Beispiel

```
static void test() {
    int x;
    Book myBook;

    println( x );
    println( myBook.title );
}
```

Tasks (Filter matched 2 of 30 items)

✓	!	Description
✗		The local variable x may not have been initialized
✗		The local variable myBook may not have been initialized

Bei *Feldern* ist es anders: Default-Initialisierung

- Ist ein *Feld* einer Klasse *nicht initialisiert*, dann weist ihm die VM abhängig von seinem Typ einen sogenannten ***Default-Wert*** zu

Beispiel

```
class Book {
    private String title;
    private int inventaryNumber;

    static void test1() {
        Book myBook = new Book();

        println( myBook.inventaryNumber );
        println( myBook.title );
    }

    void test2() {
        println( inventaryNumber );
        println( title );
    }

}
```

```
Console [<terminated> C:\app\j2sdk1.4.0\bin\javaw.exe (08.12.0
0
null
```

Default-Werte für Datentypen von Feldern

- Die Zuweisung eines Default-Wertes an Felder gilt für alle Datentypen

Typ	**Default-Wert**
boolean	false
char	‘\u0000’ (0)
byte	(byte) 0
short	(short) 0
int	0
long	0L
float	0.0f
double	0.0d
String	null
Klasse(ntyp)	null
Array(typ)	null
Aufzählungstyp	null

Beispiel mit Default-Werten

- Beim Aufruf des Konstruktors werden alle Felder des neuen Objekts der Klasse *Book* mit *Default-Werten* versehen
- Im Beispiel sind das die folgenden Werte

```
class Book {
    private String  author;           ← null
    private String  title;            ← null
    private int     yearOfPublication; ← 0
    private boolean present;          ← false
    private Date    returnDate;       ← null
    private int     inventaryNumber;  ← 0
}
```

Default- und Initialwerte

- Alternativ ist es möglich, die Default-Werte durch eigene *Initialwerte* zu ersetzen
- Das können zum Beispiel die folgenden Werte sein

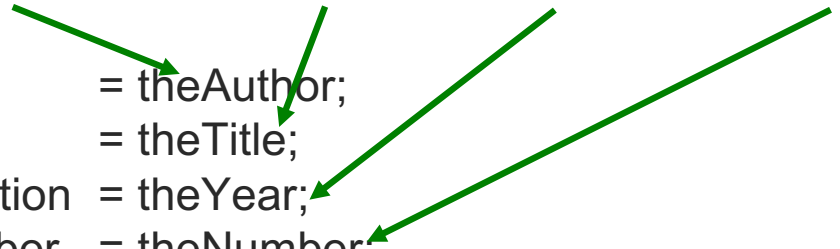
```
class Book {
    private String    author              = "";
    private String    title               = "";
    private int       yearOfPublication;          ← 0
    private boolean   present             = true;
    private Date      returnDate;                 ← null
}
```

Konstruktoren mit Parametern 1/2

- Wie Methoden können Konstruktoren mit formalen Parametern versehen werden

  Beispiel

```
Book ( String theAuthor, String theTitle, int theYear, int theNumber )
{
      author              = theAuthor;
      title               = theTitle;
      yearOfPublication   = theYear;
      inventaryNumber     = theNumber;
}
```



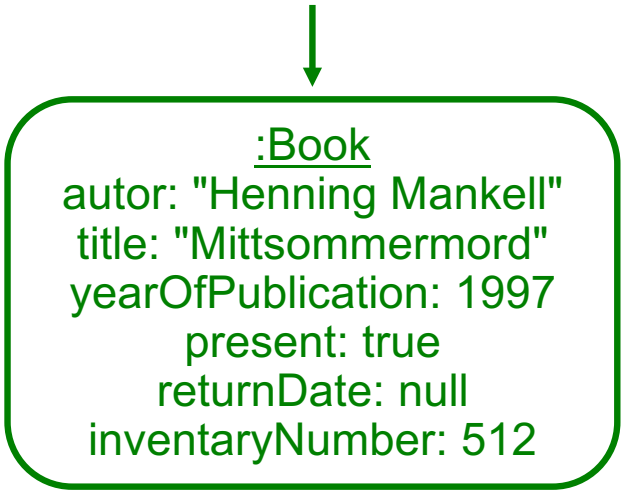
- Der Aufruf dieses Konstruktors erfolgt wie bisher mit *new* und dem Klassennamen, nur werden diesmal aktuelle Parameter übergeben

  Beispiel

```
myBook = new Book( "Henning Mankell", "Mittsommermord", 1997, 512 );
```

Konstruktoren mit Parametern 2/2

```
new Book( "Henning Mankell", "Mittsommermord", 1997, 512 );
```




```
class Book {
private String    author = "";
private String    title = "";
private int       yearOfPublication;
private boolean   present = true;
private Date      returnDate;
private int       inventaryNumber;


Book ( String theAuthor, String theTitle, int theYear, int theNumber )
{
   author              = theAuthor;
   title               = theTitle;
   yearOfPublication   = theYear;
   inventaryNumber     = theNumber;
}

}
```

Konstruktoren vs. Methoden

- Konstruktoren werden ähnlich definiert wie Methoden, aber mit folgenden Unterschieden
  1. Der Name des Konstruktors ist festgelegt:
     Der Konstruktor *heißt* immer exakt so *wie die Klasse*, die er instanziieren soll
     Beispiel:
     Heißt die Klasse „Auto“, dann heißt auch der Konstruktor „Auto“
  2. Es wird kein Ergebnistyp angegeben:
     Der Konstruktor *liefert immer ein Objekt seiner Klasse* als Ergebnis, daher kann die Angabe entfallen
     Beispiel:
     Der Konstruktor der Klasse „Auto“ liefert immer ein Objekt der Klasse „Auto“ als Ergebnis
  3. Ein Konstruktor enthält *keine* return-*Anweisung mit Ergebniswert*:
     Beim Konstruktor-Aufruf wird Speicherplatz für das neue Objekt angelegt; Anweisungen im Rumpf des Konstruktors beziehen sich auf die Felder des neu angelegten Objekts, dieses Objekt wird dann als Ergebnis des Aufrufs geliefert
     Exkurs: Wird ein *return ohne Wert* angegeben, wird der Konstruktor an dieser Stelle beendet und das erzeugte Objekt im aktuellen Zustand zurückgegeben

Bemerkungen

- Abgesehen von den genannten Einschränkungen (Name festgelegt, Ergebnistyp festgelegt, kein *return* mit Ergebniswert) können Konstruktoren alle Aufgaben übernehmen, die auch Methoden übernehmen können

  Sie *sollten* aber nach Möglichkeit *ausschließlich für die Initialisierung von Feldern* der Klasse *benutzt* werden

- Zu den sinnvollen Aufgaben eines Konstruktors gehört zum Beispiel die Initialisierung geschachtelter Strukturen (wie zum Beispiel das Datumsobjekt *returnDate* in den Büchern)

- In der Praxis kommt es darauf an, sinnvolle Initialisierungswerte für Felder vorzusehen und im Konstruktor nur Parameter für diejenigen Felder vorzusehen, die erst bei der Instanziierung bekannt sind

Beispiel für einen Konstruktor mit Parametern

- Ein Datum kann wie folgt dargestellt werden

```
class Datum {
    private int tag, monat, jahr;
    ...
}
```

- Ein sinnvoller Konstruktor für die Klasse *Datum* bietet eine komfortable Möglichkeit, ein Datumsobjekt zu erzeugen und gleichzeitig mit Werten zu versehen

```
Datum ( int derTag, int derMonat, int dasJahr ) {
    tag = derTag;
    monat = derMonat;
    jahr = dasJahr;
}
```

- Datumsobjekte können dann wie folgt erzeugt werden

```
Datum datum = new Datum( 11, 11, 2003 );
datum = new Datum( 1, 5, 1998 );
Datum geburtstag = new Datum( 18, 4, 1968 );
```

Im Überblick: der Konstruktor der *Datum*-Klasse

- Klassendefinition

```
class Datum {
    private int tag, monat, jahr;

    Datum    ( int derTag, int derMonat, int dasJahr ) {

        tag         = derTag;
        monat       = derMonat;
        jahr        = dasJahr;
    }
    ...
}
```

- Mögliche Aufrufe

```
Datum x = new Datum( 23, 3, 2003 ),
          y = new Datum( 1, 1, 2004 );
```

Problem: vernünftige Parameternamen

- Klassendefinition

```
class Datum {
    private int tag, monat, jahr;

    Datum    ( int tag, int monat, int jahr ) {

      tag   = tag;
      monat = monat;
      jahr  = jahr;
    }
    ...
}
```

unsinnige Zuweisungen

- Mögliche Aufrufe

```
Datum x = new Datum( 23, 3, 2003 ),
          y = new Datum( 1, 1, 2004 );
```

Der Konstruktor: Benutzung von *this*

- Klassendefinition

```
class Datum {
    private int tag, monat, jahr;

    Datum    ( int tag, int monat, int jahr ) {

      this.tag   = tag;
      this.monat = monat;
      this.jahr  = jahr;
    }
    ...
}
```

- Mögliche Aufrufe

```
Datum x = new Datum( 23, 3, 2003 ),
          y = new Datum( 1, 1, 2004 );
```

Bedeutung von *this*

- Klassendefinition

```
class Datum {
    private int tag, monat, jahr;

    Datum     ( int tag, int monat, int jahr ) {

        this.tag   = tag;
        this.monat = monat;
        this.jahr  = jahr;
    }
    ...
}
```

***this* sorgt für den Bezug auf Felder *dieses* (engl.: *this*) Objekts (das gerade erzeugt wird)**

- Mögliche Aufrufe

```
Datum x = new Datum( 23, 3, 2003 ),
          y = new Datum( 1, 1, 2004 );
```

Der "Bezeichner" *this* 1/3

- Der Bezeichner ***this*** ist in Java ein Schlüsselwort
- Das bedeutet, „this“ kann nicht als Variablen- oder Feld-, Klassen-, etc.-Name verwendet werden
- (Erinnerung: nicht-statische Methoden werden mit

  Objekt.*Methodenname* (aktuelle *Parameter*)

  aufgerufen)

  Innerhalb einer Methode kann mit „this“ auf das *Objekt* zugegriffen werden, mit dem die Methode aufgerufen wurde

  Ebenso kann innerhalb eines Konstruktors mit „this“ auf das Objekt zugegriffen werden, das gerade neu erstellt wird

Der "Bezeichner" *this* 2/3

- In *statischen* Methoden, d.h. in Methoden, die mit dem Schlüsselwort „static“ vereinbart sind, gibt es kein „this“, der Compiler meldet einen entsprechenden Fehler.
  Warum genau?

→Statische Methoden werden nicht mit einem Objekt aufgerufen (eigentlich: "…müssen nicht unbedingt … aufgerufen werden")!

→Also gibt es kein Objekt, auf das „this“ sich beziehen könnte!

Der "Bezeichner" *this* 3/3

- „this“ kann benutzt werden
  - um Felder der Klasse zu bezeichnen und
  - um weitere nicht-statische Methoden der Klasse aufzurufen
- Wenn der Bezug klar ist, das heißt keine Namenskonflikte zwischen lokalen Variablen oder Parametern und Feldern der Klasse existieren, *kann* „this“ *entfallen*
- Wir *dürfen* aber *immer* „this“ schreiben