Einführung in die Informatik

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Sommersemester 2014

Teil IV

Methoden in Java

Binomialkoeffizient

In einer Übungsaufgabe sollte der Binomialkoeffizient berechnet werden:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

Beispielprogramm

Möglicher Rechenweg: Berechne $n!$, $k!$, $(n-k)!$ und dividiere.

```
public class Binomial {
    public static void main(String[] param) {
        int n = IOTools.readInteger("n: ");
        int k = IOTools.readInteger("k: ");
        long nf = 1L, kf = 1L, nkf = 1L;
        for (int i = 1; i <= n; i++)
            nf *= i;
        for (int i = 1; i <= k; i++)
            kf *= i;
        for (int i = 1; i <= n-k; i++)
            nkf *= i;
        System.err.println((nf / kf / nkf));
    }
}
```

Codeduplikation

In dem Programm gibt es drei Schleifen, die die Fakultät berechnen.

- Wir müssen dreimal fast den gleichen Code schreiben.
- Wenn wir dabei einen Fehler machen, müssen wir ihn dreimal korrigieren.
- Der Code ist schwieriger zu lesen

Wie können wir die Duplikation der Schleife vermeiden?

Methoden

Man kann gemeinsamen Code in Methoden (auch Funktionen oder Prozeduren genannt) auslagern.

```
public class Binomial {
    public static long fakultaet(int n) {
        long fak = 1L;
        for (int i = 1; i <= n; i++)
            fak *= i;
        return fak;
    }
    public static void main(String[] param) {
        int n = IOTools.readInteger("n: ");
        int k = IOTools.readInteger("k: ");
        System.err.println(
           fakultaet(n) / fakultaet(k) /
              fakultaet(n - k));
    }
}
```

Syntax von Methoden

Die folgende Zeile leitet eine neue Methode ein.

```
public static long fakultaet(int n) {
```

- `public` bedeutet, dass sie von außen aufgerufen werden kann.
- `static` wird später erklärt.
- `long` ist der Typ des Rückgabewertes der Methode
- `fakultaet` ist der Name der Methode. Der kann beliebig gewählt werden.
- `int n` ist der Parameter der Methode

Die Methode nimmmt als Eingabe einen Parameter `n` vom Type `int` und liefert als Rückgabe einen Wert vom Typ `long`.

Methodenkörper

```
public static long fakultaet(int n) {
    long fak = 1L;
    for (int i = 1; i <= n; i++)
        fak *= i;
    return fak;
}
```

- Zwischen den geschweiften Klammern stehen Anweisungen.
- Eine Methode kann mehrfach aufgerufen werden. Die Anweisungen werden jedesmal neu ausgeführt.
- Jede Methode „sieht“ ihre eigenen Variablen. Die Variable `n` der Methode `fakultaet` ist nicht gleich der Variable `n` in `main`.
- Die Parameter werden auf den Wert gesetzt, der beim Aufruf angegeben wird. Z.B. setzt der Aufruf `fakultaet(n-k)` den Wert von `n` in Methode `fakultaet` auf das Ergebnis der Berechnung `n-k`.
- Der Befehl `return` beendet die Methode und setzt den Rückgabewert.

Die Methode `main`

Auch `main` ist eine Methode.

- Der Rückgabetyp ist `void`. Dieser spezielle Typ steht für keinen Wert.
- Eine Methode mit dem Rückgabetyp `void` braucht kein `return`. Man kann aber die `return`-Anweisung (ohne Wert) nutzen, um die Methode vorzeitig zu verlassen.
- Der Parameter ist `String[] param`. Wir werden später sehen, dass das für ein Feld von Zeichenketten (also beliebig viele Zeichenketten) steht.

Ausführungsreihenfolge

Wenn eine Methode aufgerufen wird, wird die aktuelle Methode unterbrochen und später weiter ausgeführt.

```
public class Binomial {
    public static long fakultaet(int n) {
        System.err.println("in fakultaet n = "+n);
        long fak = 1L;
        for (int i = 1; i <= n; i++)
            fak *= i;
        System.err.println("in fakultaet fak = "+fak);
        return fak;
    }
    public static void main(String[] param) {
        int n = IOTools.readInteger("n: ");
        int k = IOTools.readInteger("k: ");
        System.err.println("in main n = " + n);
        long b = fakultaet(n)/fakultaet(k)/fakultaet(n-k);
        System.err.println("in main n = " + n);
        System.err.println("in main b = " + b);
    }
}
```

Parameter und Rückgabewerte

Eine Methode kann (fast) beliebig viele Parameter haben, aber nur maximal einen Rückgabewert.
Zum Beispiel:

```
public static long binomial(int n, int k) {
    return fakultaet(n) / fakultaet(k) /
        fakultaet(n-k);
}
```

Call by Value

Der Parameter wird als Wert übergeben. Wenn Methoden ihre Parameter ändern hat das auf den Aufrufer keine Auswirkung:

```
public static long fakultaet(int n) {
    long fak = 1L;
    while (n > 0)
        fak *= (n--);
    return fak;
}
...
    int  n = 5;
    long f = fakultaet(n);
    System.err.println(n);
```

Hier ändert fakultaet zwar den Parameter `n`, aber das `n` des Aufrufers wird nicht geändert.

Dokumentation

Man sollte zu jeder Methode eine Dokumentation schreiben, die die Methode und ihre Parameter erklärt und die Sonderfälle erläutert.

```
/**
 * Berechnet die Fakult&auml;t von n.
 * @param n  eine nichtnegative ganze Zahl.
 *   Um &Uuml;berl&auml;ufe auszuschlie&szlig;en
 *   muss <code>n &lt;= 20</code> gelten.
 * @returns Die Fakult&auml;t von n.
 */
public static long fakultaet(int n) {
```

Man kann eingeschränkte HTML-Befehle in der Dokumentation benutzen (deshalb muss man aber auch $<$ als `<` schreiben).

Rekursive Methoden

In der Mathematik definiert man Fakultät auch oft induktiv durch

$$0! = 1 \qquad n! = n \cdot (n-1)! \text{ (falls } n > 0)$$

Hier wird bei der zweiten Definition die Fakultät von $n-1$ benutzt um die Fakultät von n zu berechnen (man nennt das Rekursion).
Das geht auch in der Programmiersprache Java:

```
public long fakultaet(int n) {
    if (n == 0)
        return 1;
    return n * fakultaet(n - 1);
}
```

Binomial als Rekursive Methode

Auch mehrfache Aufrufe sind möglich.

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n \end{cases}$$

```
public long binomial(int n, int k) {
    if (k == 0 || k == n)
        return 1;
    return binomial(n-1, k-1)
         + binomial(n-1, k);
}
```

Teil V

Referenzdatentypen

Felder (Arrays)

Felddatentypen

Die bisher vorgestellten Datentypen können nur einen Wert speichern (Ausnahme: Zeichenkette). Um mehrere Werte zu speichen gibt es Feldtypen (array types).

Feldtyp

Zu jedem Datentyp `T` gibt es einen Datentyp mit dem Namen `T[]`, der mehrere Elemente vom Typ T speichern kann. `T[]` ist ein Feldtyp.

Den Typ `String[]` haben wir schon als Parameter der `main`-Funktion gesehen. Der Parameter enthält die Kommandozeilenargumente.

Lesen von Feldern

Das folgende Programm gibt die Kommandozeilenargumente aus:

```
public class PrintArgs {
    public static void main(String[] params) {
        for (int i = 0; i < params.length; i++) {
            System.out.println("Argument "+i+"
               ist "+params[i]);
        }
    }
}
```

```
> java PrintArgs Hallo Welt
Argument 0 ist Hallo
Argument 1 ist Welt
```

Lesen von Feldern (2)

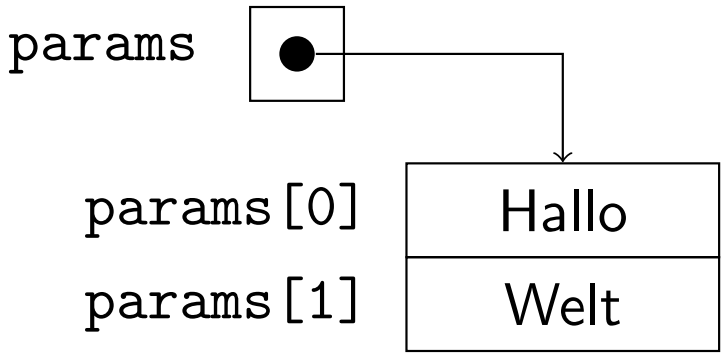
```
for (int i = 0; i < params.length; i++) {
    System.out.println("Argument " + i
                       + " ist " + params[i]);
}
```

Sei `params` eine Variable, die einen Feldtyp hat.

- `params.length` gibt die Länge des Felds (also die Anzahl der gespeicherten Werte) zurück.
- `params[i]` gibt das `i`-te Element des Feldes zurück. Man nennt `i` auch den Feldindex.
  Die Felder werden von 0 bis `params.length - 1` nummeriert.

Felder sind Referenztypen

Die Variable `params` enthält in Wirklichkeit nur einen Zeiger (auch Referenz genannt) auf das Feld. Das Feld selbst kann irgendwo im Speicher liegen.



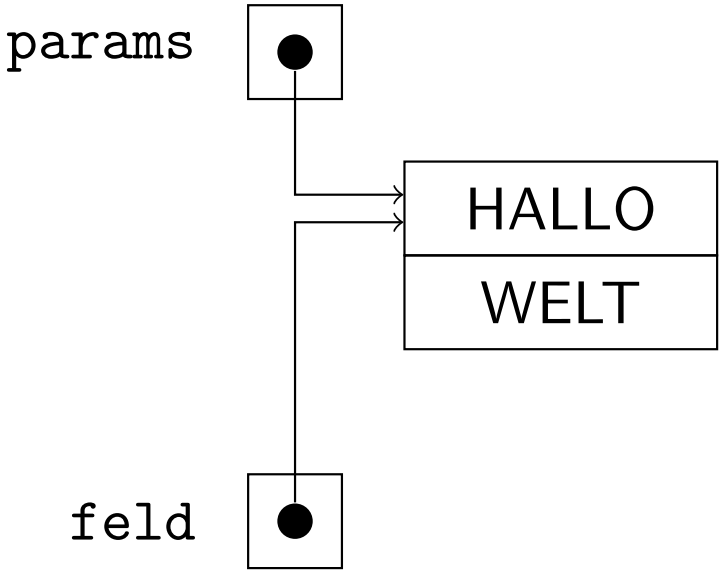
Bei einem Methodenaufruf wird nur der Zeiger übergeben, das Feld selbst wird nicht kopiert.

Schreiben auf Felder

```
public class PrintArgs {
    public static void macheGross(String[] feld){
        for (int i = 0; i < feld.length; i++)
            feld[i] = feld[i].toUpperCase();
    }
    public static void main(String[] params) {
        macheGross(params);
        for (int i = 0; i < params.length; i++)
            System.out.println(params[i]);
    }
}
```

Der Ausdruck `feld[i]` darf auch auf der linken Seite einer Zuweisung stehen. Dadurch wird der Inhalt des Feldes geändert.

Ablauf des Programms



- Beim Aufruf von `macheGross` wird der Zeiger von `params` nach `feld` kopiert.
- Durch `feld[0] = feld[0].toUpperCase()` wird der Inhalt des Feldes geändert.
- Nach dem Ende der Methode gibt es `feld` nicht mehr. Die Parameter sind aber geändert.

Erzeugen von Feldern

Felder müssen erzeugt werden und dabei wird auch die Länge des Feldes festgelegt:

```
int[] meinFeld = new int[10];
```

Der Ausdruck `new int[10]` erzeugt ein Feld mit exakt 10 Elementen, die jeweils ein `int` speichern können. Der Zeiger auf dieses Feld wird der Variable `meinFeld` zugewiesen.

- Die Länge des Feldes kann nicht geändert werden, allerdings kan man `meinFeld` ein neues Feld mit zum Beispiel mehr Elementen zuweisen.
- Wenn man ein Element lesen will, dass nicht vorhanden ist (zu großer oder negativer Feldindex), stürzt das Programm mit einer Fehlermeldung ab (ArrayIndexOutOfBoundsException).
- Initial werden alle Elemente auf 0 initialisiert. Referenztypen werden auf den Wert `null` initialisiert.

Nullreferenz

Alle Referenztypen (also auch Felder) können in Java den Wert `null` haben. Dieser steht für uninitialisiert.

```
int[] meinFeld = null;
```

- Der Zeiger `null` zeigt auf einen verbotenen Speicherbereich an dem kein Feld liegt.
- Jeder Zugriff auf einen Index oder auf die Länge des Feldes, das den Wert `null` hat, führt zu einem Programmabsturz (NullPointerException).

Syntax für Feldzugriffe

Syntax

$$\textit{Ausdruck} ::= \textit{Ausdruck}\,[\textit{Ausdruck}]$$

$$\textit{LValue} ::= \textit{Ausdruck}\,[\textit{Ausdruck}]$$

Wenn `feld` ein Ausdruck vom Typ `T[]` und `i` ein Ausdruck vom Typ `int` ist, dann ist `feld[i]` sowohl ein *Ausdruck*, als auch ein *LValue* vom Typ `T`.

- In beiden Fällen werden zunächst `feld` und `i` ausgewertet. Das liefert einen Zeiger auf ein Feld und eine Zahl (den Feldindex).
- Als Ausdruck (Lesezugriff) hat `feld[i]` den Wert des `i`-ten Eintrags in diesem Feld.
- Als LValue (links von einer Zuweisung) wird dem `i`-ten Eintrag des Felds der Wert auf der rechten Seite zugewiesen.
- Wenn `feld` zu `null` auswertet, bricht in beiden Fällen das Programm mit einem Fehler ab (NullPointerException). Ebenso, wenn es keinen `i`-ten Eintrag gibt (ArrayIndexOutOfBoundsException).

Felder und Schleifen

Oft möchte man Code für alle Einträge in einem Feld ausführen, z.B.

```
for (int i = 0; i < params.length; i++)
    System.out.println(params[i]);
```

Java hat für diesen Fall noch eine zweite Syntax für die For-Schleife:

```
for (String param : params)
    System.out.println(param);
```

Hier wird eine Variable `param` deklariert und dieser nach und nach jeder Eintrag des Feldes `params` zugewiesen. Jedesmal wird dann der Schleifenkörper ausgeführt.

Allerdings gibt es *keine* Abkürzung für

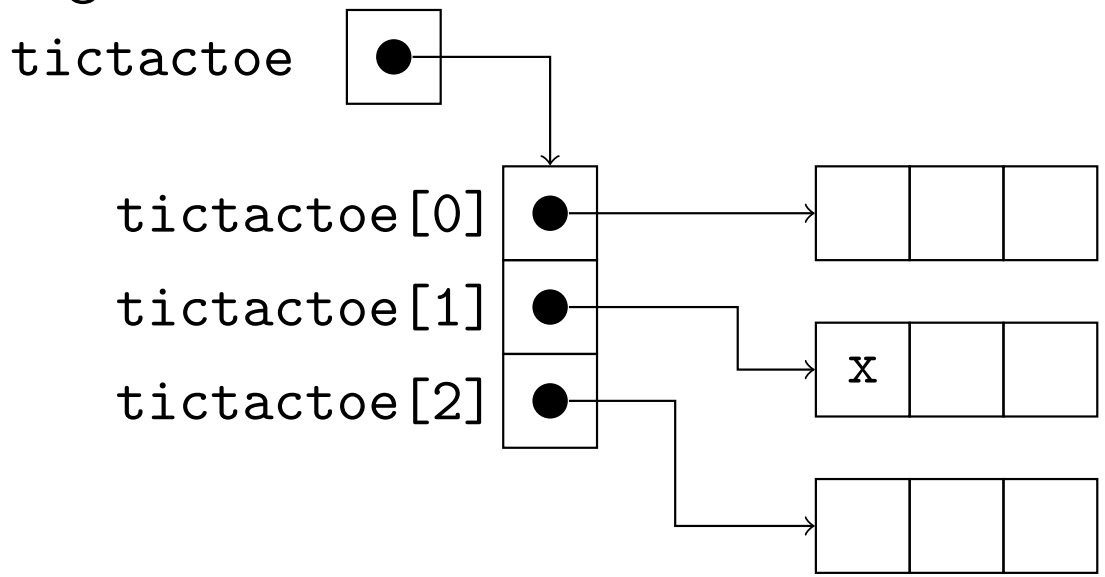
```
for (int i = 0; i < feld.length; i++)
    feld[i] = feld[i].toUpperCase();
```

Mehrdimensionale Felder

Man kann Feldtypen auch schachteln, z.B.

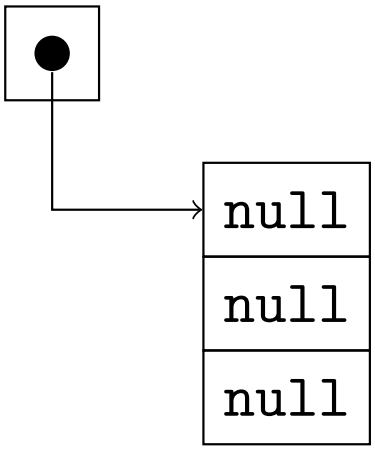
```
char[][] tictactoe = new char[3][3];
tictactoe[1][0] = 'x';
```

Hierbei ist `char[][]` ein Feld, das Werte vom Typ `char[]` speichert, also Zeiger auf `char`-Felder.



Erzeugen mehrdimensionaler Felder (1)

Der Ausdruck `new char[3][]` erzeugt ein Array mit drei Elementen vom Typ `char[]`, die mit `null` gefüllt werden:

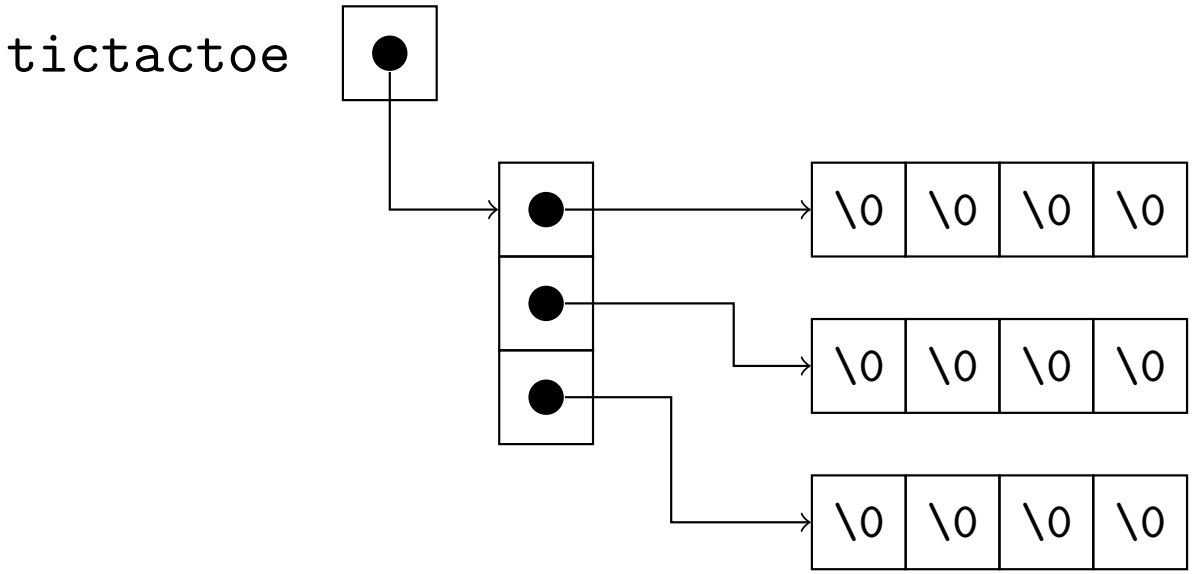


Achtung

Logisch wäre zwar `new char[][3]`, denn `char[]` ist der Typ der Argumente. Das ist aber syntaktisch falsch. Bei `new` entspricht die Reihenfolge der eckigen Klammern der Reihenfolge beim Feldzugriff.

Erzeugen mehrdimensionaler Felder (2)

Der Ausdruck `new char[3][4]` erzeugt ein Array mit drei Elementen vom Typ `char[]`, die mit neuen Arrays mit jeweils vier Elementen vom Typ `char` initialisiert werden.

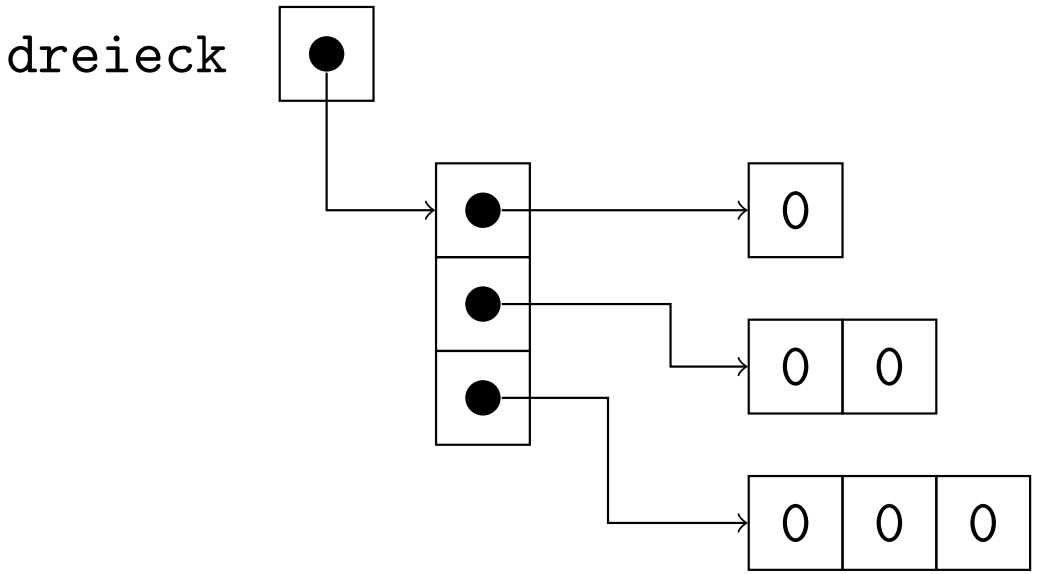


Der Ausdruck `new char[][4]` ist syntaktisch nicht erlaubt.

Erzeugen mehrdimensionaler Felder (3)

Bei mehrdimensionalen Feldern, gibt es aber keinen Grund, warum die Unterfelder gleich lang sein müsssen.

```
int[][] dreieck = new int[3][];
for (int i = 0; i < 3; i++)
    dreieck[i] = new int[i+1];
```



Klassen

Klassen

Ein weiterer Referenztyp sind Klassen. Damit kann man mehrere Werte mit unterschiedlichem Typ zusammenfassen.

```
public class Adresse {
    public String vorname;
    public String nachname;
    public String strasse;
    public int hausnummer;
    public int postleitzahl;
    public String ort;
}
```

- Der obige Java-Code deklariert einen neue Klasse (einen Typ) mit dem Namen `Adresse`, die aus mehreren Komponenten `vorname`, `nachname`, usw. besteht.
- Eine Instanz oder Objekt von diesem Typ speichert mehrere Werte, einen für jede Komponente.
- Das Schlüsselwort `public` erlaubt es jedem, die Komponente zu lesen und zu schreiben.

Wohin kann man Klassen schreiben?

- Die Klasse Adresse kann in einer Datei mit dem Namen `Adresse.java` gespeichert werden.
- Alternativ kann man auch innere Klassen verschachtelt in einer Klasse definieren:

```
public class Adressverwaltung {
    public static class Adresse {
        ...
    }

    public static void main(String[] param) {
        ...
    }
}
```

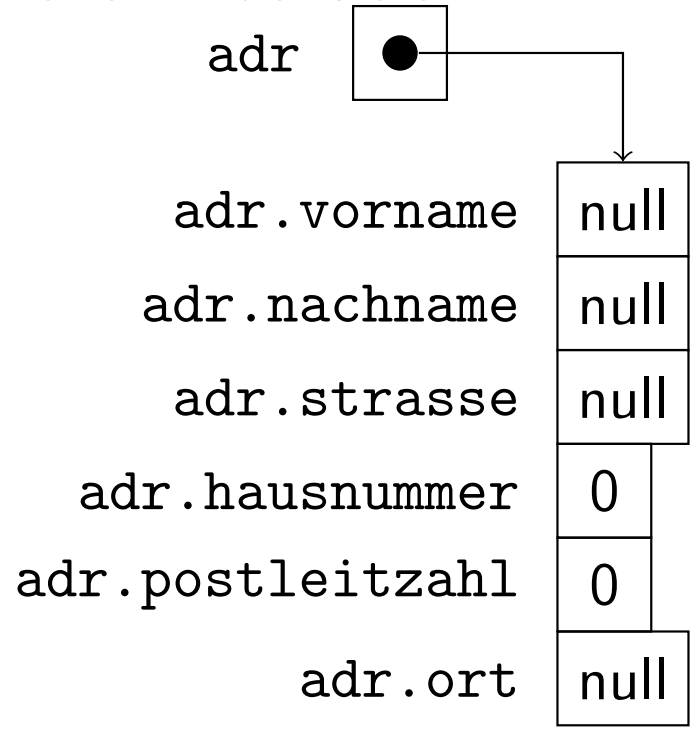
Das sollte man aber nur machen, wenn die Klasse nur von der Adressverwaltung gebraucht wird.

Objekt erzeugen

Die folgende Anweisung erzeugt ein neues Objekt vom Typ Adresse und weist sie der Variablen adr zu.

```
Adresse adr = new Adresse();
```

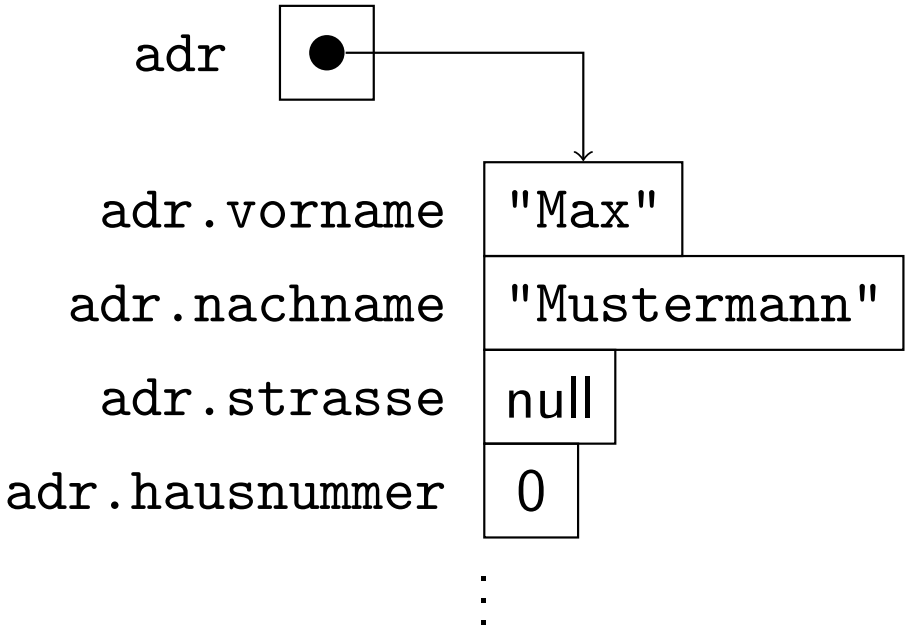
Auch hier wir in adr eine Referenz auf das eigentliche Objekt geschrieben. Das Objekt selbst kann irgendwo im Speicher liegen. Das Objekt wird mit Nullen initialisiert.



Objekte schreiben

Mit der Punkt-Notation kann auf die Komponenten eines Objekts zugegriffen werden.

```
Adresse adr = new Adresse();
adr.vorname = "Max";
adr.nachname = "Mustermann";
System.out.println(adr.vorname + " " +
                   adr.nachname);
```



Syntax der Punktnotation

Syntax

$$\textit{Ausdruck} ::= \textit{Ausdruck} \,.\, \textit{Bezeichner}$$

$$\textit{LValue} ::= \textit{Ausdruck} \,.\, \textit{Bezeichner}$$

Wenn `adr` ein Ausdruck vom Typ `Adresse` ist und `vorname` ein Komponente vom Typ `Adresse`, dann ist `adr.vorname` sowohl ein *Ausdruck*, als auch ein *LValue*.

- In beiden Fällen wird zunächst `adr` ausgewertet. Das liefert einen Zeiger auf ein Objekt.
- Als Ausdruck (Lesezugriff) hat `adr.vorname` den Wert der Komponente `vorname` in diesem Objekt.
- Als LValue (links von einer Zuweisung) wird dem Komponente `vorname` des Objekts der Wert auf der rechten Seite zugewiesen.
- Wenn `adr` zu `null` auswertet, bricht in beiden Fällen das Programm mit einem Fehler ab (NullPointerException).

Felder und Klassen

Man kann Felder und Klassen auf verschiedene Weise kombinieren.

- Man kann ein Feld von Objekten erzeugen:

```
Adresse[] adressListe = new Adresse[10];
adressListe[0] = new Adresse();
```

- man kann in einer Klasse Felder oder andere Klassen als Komponenten haben.

```
public class Brief {
    public String[] anhaenge;
    public Adresse absender;
    public Adresse empfaenger;
    ...
}
```