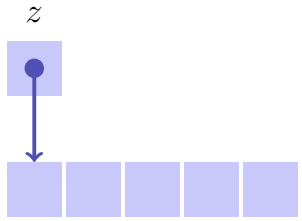
6. Java Arrays und Strings

Allokation, Referenzen, Elementzugriff, Mehrdimensionale Arrays, Strings, Stringvergleiche

Arrays

Arrayvariable deklarieren: `int[] z;`

Array erzeugen: `z = new int[5];`



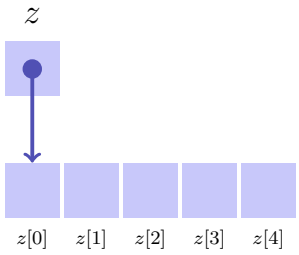
z ist eine *Referenz* auf die Arraydaten,

- aber erst nach der Zuweisung zu den erstellten Daten
- sonst zeigt es nirgendwo hin: `null`.

Arrays

`int[] z;`

`z = new int[5];`



Elemente werden indiziert. Index beginnt bei 0 und endet bei Arraygrösse -1.

Elementzugriff: `name[index]`

Arrays sind dynamische Objekte

Arrays sind grundsätzlich *dynamisch* erzeugt.

```
int[] b;
b = new int[10]; // 10 Elemente mit indizes 0...9
...
b = new int[20]; // kann neu zugewiesen werden
```

Grösse eines Arrays kann also zur Laufzeit festgelegt werden.
Ein Array wächst jedoch nicht automatisch!

Arrays sind nicht primitiv

Arrays tragen *Metadaten* mit sich herum:

```
int sq = new int[7];
for (int i = 0; i< sq.length; ++i){
    sq[i] = i * i;
}
sq[8] = 64;
```
⟵ java.lang.ArrayIndexOutOfBoundsException!

... auch über Funktionengrenzen hinweg:

```
static void print(int[] a){
    for (int i = 0; i< a.length; ++i){
        System.out.println("a[" + i + "]=" + a[i]);
    }
}
```


Arrayzuweisungen

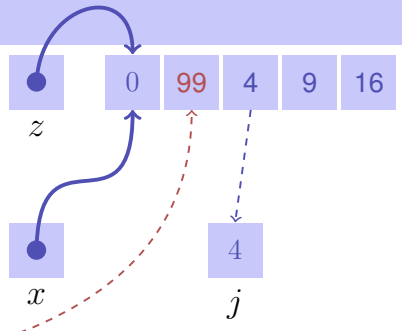
```
int[] z = new int[5];

for (int i=0; i<z.length; ++i)
    z[i] = i*i;

int[] x = z;

int j = x[2];

x[1] = 99;
```



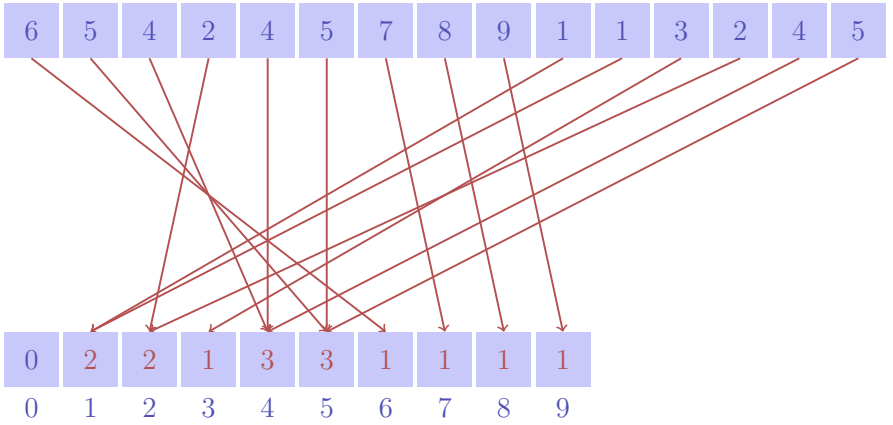
Bei der Zuweisung von Arrays *wird die Referenz kopiert*, nicht die Daten!

Beispiel

Annahme: Ein (unsortiertes) Array x enthalte nur die Zahlen $0, \ldots, 9$.

Aufgabe: Schreibe ein effizientes Programm, welches für jede solche Zahl ausgibt, wie oft sie in x vorkommt.

Idee



Code

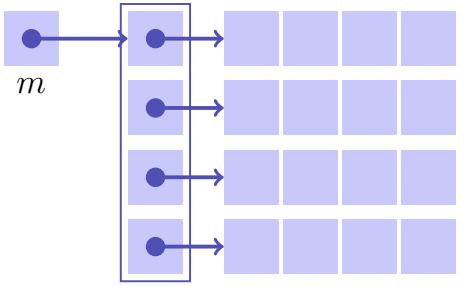
```
public class CountNumbers {
  public static void main(String [] args) {
    int [] zahlen = {5, 4, 2, 4, 5, 7, 8, 9, 1, 1, 3, 2, 4, 5};
    int [] index = new int [10];

    for (int i = 0; i < zahlen.length ; i++) {
      index [zahlen[i]]++;
    }

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


Mehrdimensionale Arrays

```
double[][] m = new double[4][4];
```



Mehrdimensionale Arrays

```
double[][] matrix=new double[4][4];

// Einheitsmatrix
for (int r=0; r < matrix.length; ++r){
  for (int c=0; c < matrix[r].length; ++c){
    if (r==c)
        matrix[r][c] = 1;
    else
        matrix[r][c] = 0;
  }
}
```


Mehrdimensionale Arrays

Ein zweidimensionales Array ist ein Array von Referenzen auf eindimensionale Arrays. Also geht auch das:

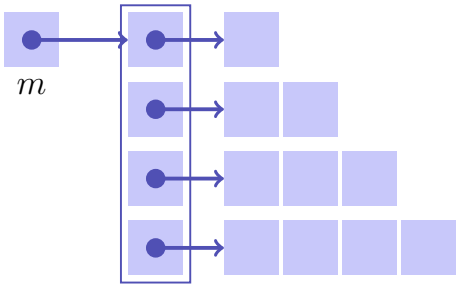
```
static void printVector (double[] vector){
    for (int c = 0; c<vector.length; ++c){
        System.out.print(vector[c] + " ");
    }
}

static void printMatrix (double[][] matrix){
    for (int r = 0; r<matrix.length; ++r){
        printVector(matrix[r]);
        System.out.println();
    }
}
```

Mehrdimensionale Arrays

Es geht sogar das:

```
double[][] m = new double[5][];
for (int r = 0; r < m.length; ++r)
  m[r] = new double[r+1];
```



Array Vergleiche

Erneut aufgepasst: Arrays sind Referenzen!!

```
double[] x = {1,2,3};
double[] y = x;
double[] z = {1,2,3};

if (y == x) {...} // y==x ist true
if (z == x) {...} // z==x ist false!
```

Für Kenner:

```
if (z.equals(x)) {...}       // z.equals(x) ist auch false !!
if (Arrays.equals(x,z)) {...} // Arrays.equals(x,z) ist true.
```

Vorsicht bei `Arrays.equals` bei mehrdimensionalen Arrays! (Was wird wohl geprüft?)

Strings

String: ein Objekt, welches Zeichenketten speichert.

```
String name = "Informatik";
String hochschule = "ETH";
String vorlesung = name + " an der " + hochschule;
int x = 3;
int y = 5;
String koordinaten = "(" + x + "," + y + ")"; // "(3,5)"
```


Strings

Vorsicht, Evaluationsreihenfolge beachten:

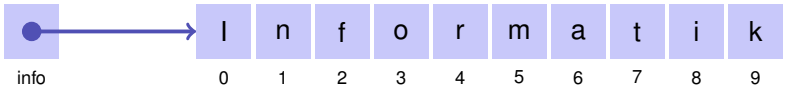
```
int x = 3;
int y = 5;
String s1 = x+y+"X";  // s1 = "8X"
String s2 = "X"+x+y+""; // s2 = "X35"
```

Characters

Elemente eines Strings können auch per Index gelesen (nicht aber geschrieben) werden:

```
String info = "Informatik";
char c = info.charAt(3); // c = 'o'
```

Strings sind auch Referenzen!



Stringvergleiche

Der Vergleich mit '==' vergleicht Referenzen, nicht den Inhalt!

```
Scanner scanner = new Scanner(System.in);
String n1 = scanner.next();
String n2 = scanner.next();
if (n1 == n2)
  System.out.println(n1 + "==" +n2);
else
  System.out.println(n1 + "!=" + n2);
}
```

Eingabe: Info Info

Ausgabe: Info != Info

Stringvergleiche

Der Vergleich mit 'equals' vergleicht den Inhalt![6]

```
Scanner scanner = new Scanner(System.in);
String n1 = scanner.next();
String n2 = scanner.next();
if (n1.equals(n2))
  System.out.println(n1 + "==" +n2);
else
  System.out.println(n1 + "!=" + n2);
}
```

Eingabe: Info Info

Ausgabe: Info == Info

[6] Bei Arrays geht das nicht!

7. MCMC Simulation

Simulieren eines fairen Würfels, Simulieren eines unfairen Würfels, Wettersimulation, Markovketten informell.

Würfel simulieren

Gegeben: Simulation uniformverteilter Zufallsvariable `Math.Random()` $\in [0,1)$.

Gesucht: Simulation eines *fairen* Würfels

Würfel simulieren

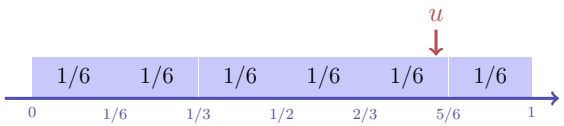
`Math.Random()` gibt ein $U \in [0,1)$ zurück mit

$$\mathbb{P}(U \in [l,r)) = r - l.$$

`Dice()` soll ein $Y \in \{1,\ldots,6\}$ zurückgeben, so dass

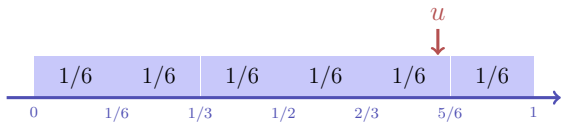
$\mathbb{P}(Y = k) = 1/6$ für jedes $k \in \{1,\ldots,6\}$

Umständlich, aber korrekt




```
static int Dice(){
  double u = Math.random();
  if (u<1.0/6) return 1;
  else if (u<1.0/3) return 2;
  else if (u<1.0/2) return 3;
  else if (u<2.0/3) return 4;
  else if (u<5.0/6) return 5;
  else return 6;
}
```


Einfacher

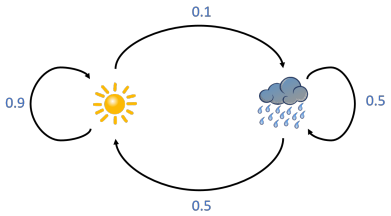



```
static int Dice(){
  double u = Math.random();
  return (int)(u*6+1);
}
```

(Zu) simples Wettermodell

- Wenn es heute sonnig ist, dann auch morgen mit Wahrscheinlichkeit 90%.
- Wenn es heute regnet, dann auch morgen zu 50%.
- Frage: Wie oft ist es auf lange Sicht sonnig?

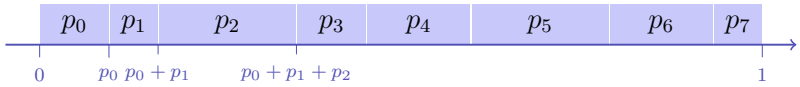
Wir simulieren das!



Unfair würfeln

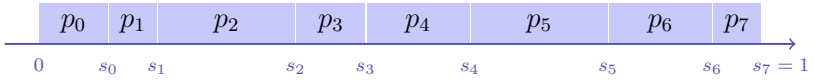
Für obiges Beispiel benötigen wir eigentlich nur eine unfaire Münze. Wir betrachten aber sofort das generelle Problem:

Gegeben: Wahrscheinlichkeitsvektor $p = (p_0, \ldots, p_{n-1})$ mit $\sum_{i=0}^{n-1} p_i = 1$ und $p_i \geq 0$ $(0 \leq i < n)$.



Gesucht: `Sample(p)` soll ein j $(0 \leq j < n)$ zurückgeben mit Wahrscheinlichkeit p_j.

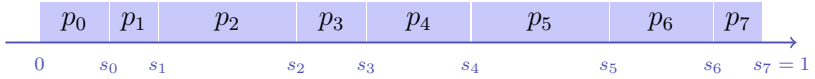
Unfair würfeln




```
static int Sample(double[] p){
  double u = Math.random();
  if (u<p[0]) return 0;
  if (u<p[0]+p[1]) return 1;
  if (u<p[0]+p[1]+p[2]) return 2;
  if (u<p[0]+p[1]+p[2]+p[3]) return 3;
  ...
}
```

Zu umständlich: wir brauchen eine Schleife!

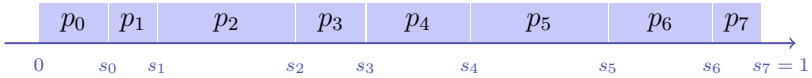
Ein kleiner Trick




```
static int Sample(double[] p){
  double u = Math.random();
  if (u<p[0]) return 0;
  u -= p[0];
  if (u<p[1]) return 1;
  u -= p[1];
  if (u<p[2]) return 2;
  ...
}
```

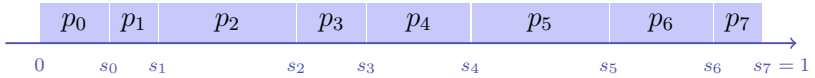
So müssen wir Summe der p_i nicht mitrechnen.

In eine Schleife




```
static int Sample(double[] p){
  double u = Math.random();
  for (int k = 0; k < p.length−1; ++k){
    if (u<p[k]) return k;
    u −= p[k];
  }
  return p.length−1;
}
```

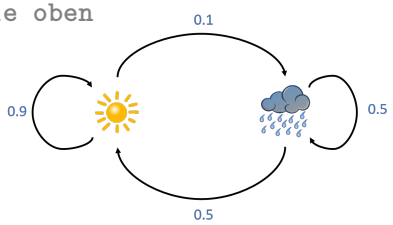

Noch kompakter




```
static int Sample(double[] p){
  double u = Math.random();
  int k=0;
  while (k < p.length && u>0){
    u −= p[k++];
  }
  return k−1;
}
```

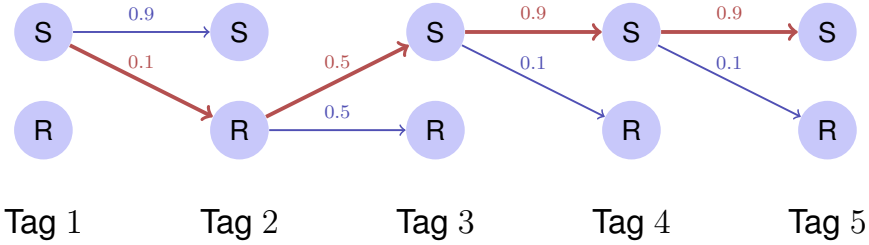

Wir reden wieder nur über das Wetter

```
static int Sample(double[] p){...} // wie oben
```



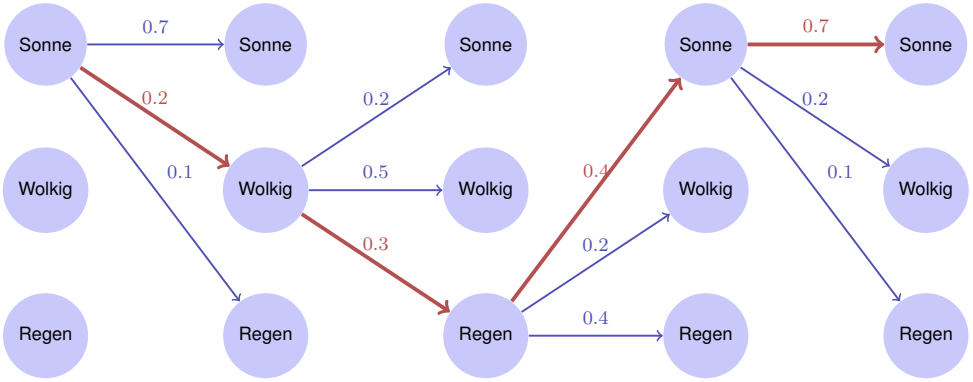

```
public static void main(){
  double[][] P = {{0.9,0.1}, {0.5,0.5}};
  int sonnig = 0;
  int wetter = 0; // sonnig
  for (int i = 0; i<365; ++i){
        wetter = Sample(P[wetter]);
        if (wetter == 0)
                sonnig++;
  }
  System.out.println("Sonnige Tage: " + sonnig);
}
```


Was haben wir mit dem Wetter gemacht?



Wetter hängt immer nur vom Vortageswetter ab: Simulation einer *Markov-Kette*.

Erweitertes Modell



Wahrscheinlichkeitsmatrix

	Sonne	Wolken	Regen
Sonne	0.7	0.2	0.1
Wolken	0.2	0.5	0.3
Regen	0.4	0.2	0.4

Zeilen = Wahrscheinlichkeitsvektoren

Fact – ohne jede Vertiefung

Wenn man eine Markov-Kette simuliert und die zugehörige Wahrscheinlichkeitsmatrix nur Einträge $\neq 0$ enthält, dann konvergiert der Vektor, welcher die relative Anzahl der besuchten Zustände ("sonnig" / "wolkig" / "regen") enthält, gegen einen Eigenvektor der Wahrscheinlichkeitsmatrix.

Anwendungsbeispiele

- Häufigkeit von Webseitenverlinkung (Google's pagerank: Bewertung von Webseiten).
- Häufigkeit von Buchstaben / Wörterkombinationen: Generierung und Analyse von Texten. (Spracherkennung)
- MCMC (Markov Chain Monte Carlo) Erzeugung von Samples auf sehr hochdimensionalen Zustandsraum. (Bildverarbeitung: Mustersynthese, Bildsegmentierung).
- Spezialfall Simulated Annealing: Probabilistische Verfahren zur Minimierung von Funktionalen. Beispiel Travelling Salesman.

8. Java Methoden

Methoden, Deklaration, Signatur, Kontrollfluss, Pass by Value

Methoden

Code-Fragmente können zu Methoden geformt werden

Vorteile:

- Einmal definieren – mehrmals verwenden/aufrufen
- Übersichtlicherer Code, einfacher zu verstehen
- Code in Methoden kann separat getestet werden

Beispiel Keksrechner

```
public class Keksrechner {

  public static void main(String[] args){
    Scanner input = new Scanner(System.in);

    System.out.print("Kinder: ");
    int kinder = input.nextInt();

    System.out.print("Kekse: ");
    int kekse = input.nextInt();

    System.out.println("Jedes Kind kriegt " + kekse / kinder + " Kekse");
    System.out.println("Papa kriegt " + kekse % kinder + " Kekse");
  }
}
```


Keksrechner - Zusätzliche Anforderung

Wir wollen sicherstellen, dass `kinder` positiv ist und jedes Kind mindestens einen Keks kriegt. $\Rightarrow$ *Eingabe prüfen!*

Keksrechner - Eingabeprüfung

Aus ...

```
System.out.print("Kinder: ");
int kinder = input.nextInt();
```

... wird demnach:

```
int kinder;
do {
    System.out.print("Kinder: ");
    kinder = input.nextInt();
    if (kinder < 1){
        System.out.println("Wert zu klein. Mindestens " + 1);
    }
} while (kinder < 1 );
```

Analog dazu müssen wir prüfen, dass `kekse >= kinder` ist.

Keksrechner - Es wird unübersichtlich

```
public class Keksrechner {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int kinder;
    do {
        System.out.print("Kinder: ");
        kinder = input.nextInt();
        if (kinder < 1){
            System.out.println("Wert zu klein. Mindestens " + 1);
        }
    } while (kinder < 1 );
    int kekse;
    do {
        System.out.print("Kekse: ");
        kekse = input.nextInt();
        if (kekse < kinder){
            System.out.println("Wert zu klein. Mindestens " + kinder);
        }
    } while (kekse < kinder);
    System.out.println("Jedes Kind kriegt " + kekse / kinder + " Kekse");
    System.out.println("Papa kriegt " + kekse % kinder + " Kekse");
  }
}
```

Anzahl Kinder einlesen und prüfen

Anzahl Kekse einlesen und prüfen

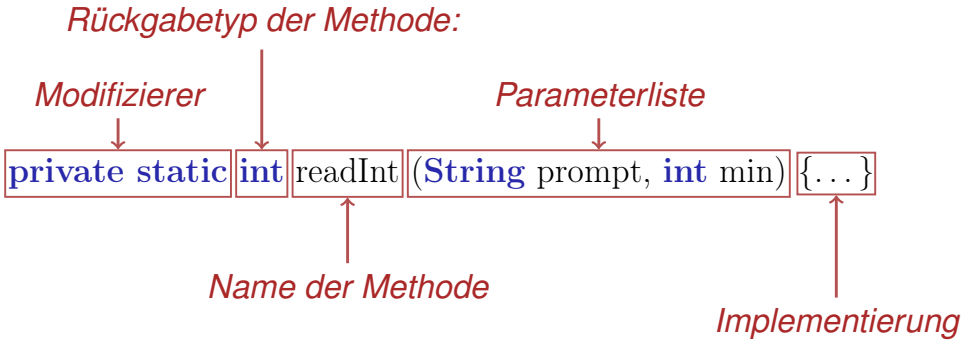
Keksrechner - Erkenntnisse

- Die beiden Code-Fragmente sind *fast identisch*
- Folgende Aspekte sind unterschiedlich:
  - Der Prompt, also `"Kinder: "` vs. `"Kekse: "`
  - Das Minimum, also "`1`" vs. "`kinder`"

Wir können das Code-Fragment in eine Methode auslagern und somit *wiederverwenden*.

Dabei müssen wir die unterschiedlichen Aspekte *parametrisieren*.

Methodendeklaration



Methodendeklaration

- **Modifizierer**: Werden später behandelt
- **Rückgabetyp**: Datentyp des Rückgabewertes. Falls die Methode keinen Wert zurückliefert, ist dieser Typ `void`.
- **Name**: Ein gültiger Name. Sollte mit Kleinbuchstaben anfangen.
- **Parameterliste** Mit rundern Klammern umgebene Liste von Parametern, deklariert mit Datentyp und Name. Parameter werden beim Aufruf der Methode gesetzt und können dann wie lokale Variablen verwendet werden.
- **Implementierung**: Der Code, welcher ausgeführt wird wenn die Methode aufgerufen wird.

Methodensignatur

```
private static int readInt (String prompt, int min) { ... }
```

Signatur der Methode

- Signatur ist eindeutig innerhalb einer Klasse
- Es ist also möglich, mehrere Methoden mit gleichem Namen aber unterschiedlichen Parameter-Anzahl und -Typen zu haben – *aber nicht empfohlen!*
- Rückgabetyp ist nicht Teil der Signatur! Es ist nicht möglich, mehrere Methoden zu haben, die sich nur im Rückgabetyp unterscheiden.

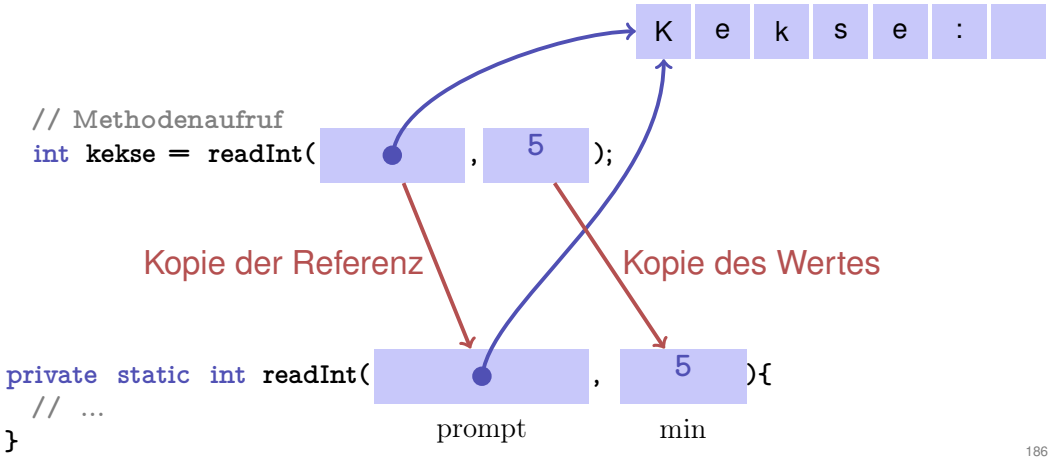
Methodenaufruf - Pass by Value

- Ein Methodenaufruf ist ein Ausdruck, dessen Wert falls vorhanden der Rückgabewert der Methode ist.
- In Java gilt immer die *Pass by Value* Semantik.

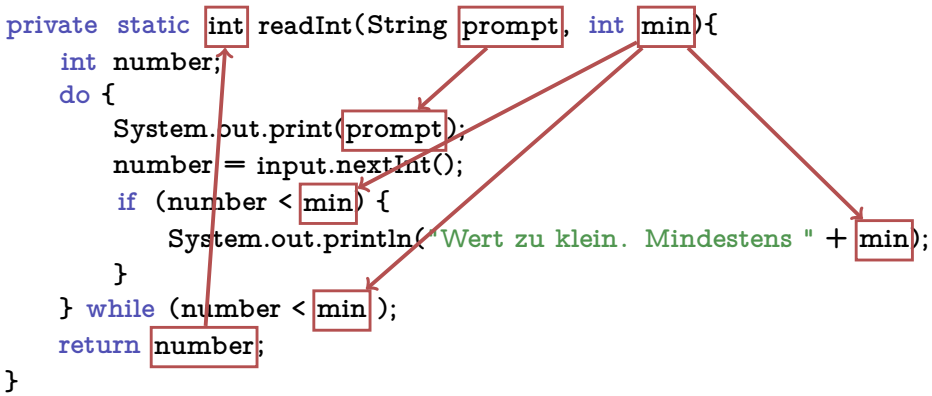
Pass by Value bedeutet: Argumentwerte werden beim Methodenaufruf in die Parameter *kopiert*.

Dies entspricht demselben Prinzip wie die Wertzuweisung an eine Variable.

Methodenaufruf - Pass by Value



Zurück zum Beispiel - Methode `readInt`



Rückgabewerte von Methoden

Es gibt zwei Fälle

- **Rückgabetyp** $=$ `void`: Die Auswertung der Methode *kann* mittels der Anweisung `return` beendet werden.
- **Rückgabetyp** $\neq$ `void`: Die Auswertung der Methode *muss* mittels der Anweisung "`return` Wert" beendet werden. Der Wert wird dann an die aufrufende Methode zurückgegeben.

Wichtig: Im zweiten Fall muss *jeder* mögliche endliche Ausführungspfad eine "`return`" Anweisung enthalten.

Rückgabewerte auf allen Ausführungspfaden

Richtig:

```
private static int readInt(String prompt, int min){
    int number;
    do {
        System.out.print(prompt);
        number = input.nextInt();
        if (number < min) {
            System.out.println("Wert zu klein. Mindestens " + min);
        }
    } while (number < min );
    return number;
}
```

Wird garantiert immer erreicht, falls Methode terminiert

Rückgabewerte auf allen Ausführungspfaden

"Inhaltlich" identische Lösung:

```
private static int readInt(String prompt, int min){
    int number;
    while (true) {
        System.out.print(prompt);
        number = input.nextInt();
        if (number >= min) return number;
        System.out.println("Wert zu klein. Mindestens " + min);
    };
}
```

Compiler versteht die Endlosschleife und akzeptiert, dass das "return" nur hier vorkommt.

Rückgabewerte auf allen Ausführungspfaden

"Inhaltlich" identische Lösung:

```
private static int readInt(String prompt, int min){
    int number;
    do {
        System.out.print(prompt);
        number = input.nextInt();
        if (number >= min) return number;
        System.out.println("Wert zu klein. Mindestens " + min);
    } while (number < min);
    return 0
}
```

Erfüllt: ein "return" auf diesem Pfad. Tatsächlich kann dies natürlich nie erreicht werden! (...warum?)

Keksrechner - Jetzt übersichtlicher

```
public class Keksrechner {

  private static Scanner input;

  public static void main(String[] args) {
    input = new Scanner(System.in);
    int kinder = readInt("Kinder: ", 1);
    int kekse = readInt("Kekse: ", kinder);
    System.out.println("Jedes Kind kriegt " + kekse/kinder + " Kekse");
    System.out.println("Papa kriegt " + kekse % kinder + " Kekse");
  }

  private static int readInt(String prompt, int min){
    // ... siehe vorige Slide
    return number;
  }
```

Klassenvariable: in allen Methoden zugänglich

Methodendeklaration - Javadoc

```
/**
 * Reads an integer value from the console input stream.
 * The radix of the input is always considered 10. This is
 * a blocking operation.
 *
 * @param prompt The prompt (without the prompt sign itself)
 *               to show before reading from the input stream
 * @param min    The smallest allowed integer
 * @return       The parsed integer value from the console
 */
private static int readInt (String prompt, int min) {...}
```


Methodendeklaration - Javadoc

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static int	readInt(java.lang.String prompt, int min) Reads an integer value from the console input stream.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methodendeklaration - Javadoc

Method Detail

readInt

```
public static int readInt(java.lang.String prompt,
                          int min)
                   throws ReadException
```

Reads an integer value from the console input stream. The radix of the input is always considered 10. This is a blocking operation.

Parameters:

`prompt - The prompt (without the prompt sign itself) to show before reading from the input stream`

`min - The smallest allowed integer`

Returns:

`The parsed integer value from the console`

Throws:

`ReadException - If either no number was read or the number was smaller than `min'`