Lektion 7

mathematische Methoden (Kosinus)
Kontrollstruktur switch
Arrays
Minsort

Methode zur Berechnung des Cosinus

- Der Cosinus lässt sich durch folgende Reihe berechnen:

$$cos\colon \mathbb{R} \to [-1,1]$$

$$cos(x) \coloneqq \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$$

- Welche Parameter und welchen Rückgabewert hat die Methode?

```
public static double cos(double x)
```

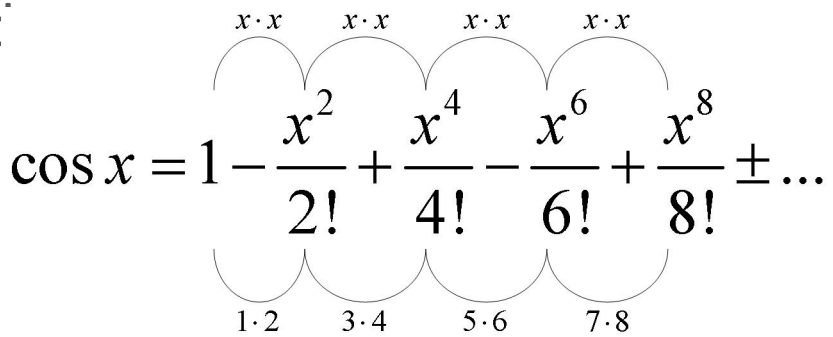
- Wie sehen die ersten fünf Glieder aus?

Methode zur Berechnung des Cosinus

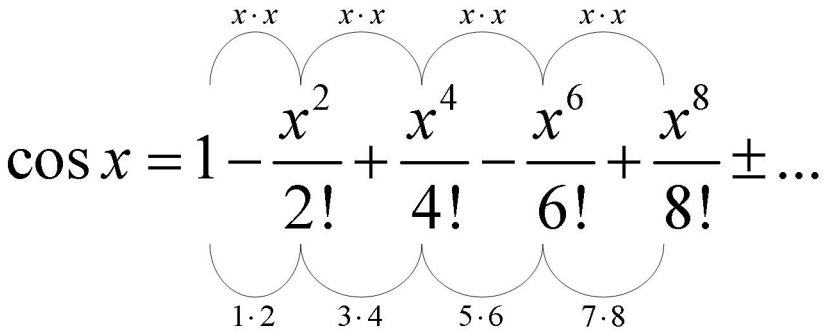
- Die ersten 5 Reihenglieder sehen wie folgt aus:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \mp \cdots$$

- Wie kann ein Algorithmus (eine Rechenvorschrift) aussehen, die den Cosinus berechnet?
- Wir schauen zunächst, wie sich Zähler und Nenner in jedem Schritt verändern:



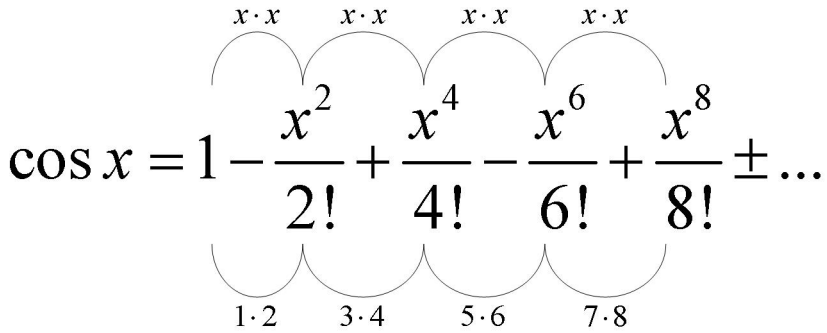
Methode zur Berechnung des Cosinus




```
public static double cos(double x)
{
   double zaehler = 1.0;
   double nenner = 1.0;
   double summe = 1.0;
```

Wir belegen den ersten Summanden vor!

Methode zur Berechnung des Cosinus

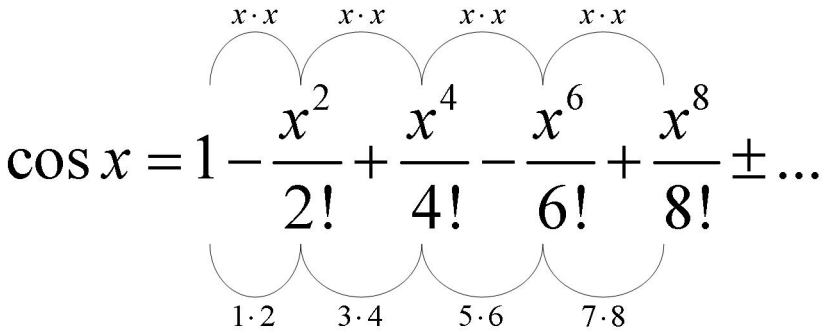



```
public static double cos(double x)
{
   double zaehler = 1.0;
   double nenner = 1.0;
   double summe = 1.0;

   for (int i = 2;          ;      )
   {
```

Wir belegen den ersten Summanden vor!

Starten die Schleife beim 2. Summanden

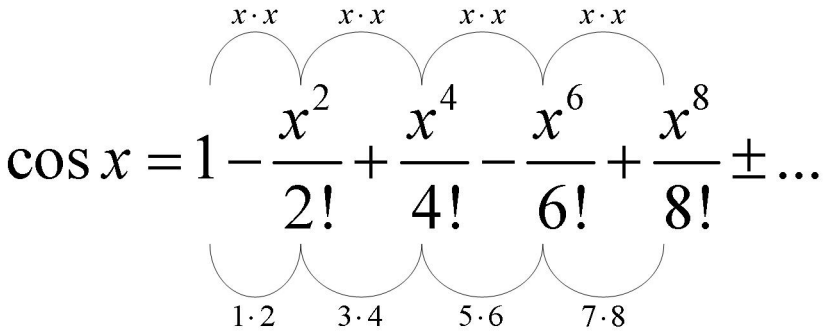
Methode zur Berechnung des Cosinus




```
public static double cos(double x)
{
  double zaehler = 1.0;
  double nenner = 1.0;
  double summe = 1.0;

  for (int i = 2;         ;      )
  {
    zaehler = zaehler * x * x * (-1);
```

Der Zähler berechnet sich, indem man den alten Zähler mit x^2 multipliziert (und das Vorzeichen ändert).

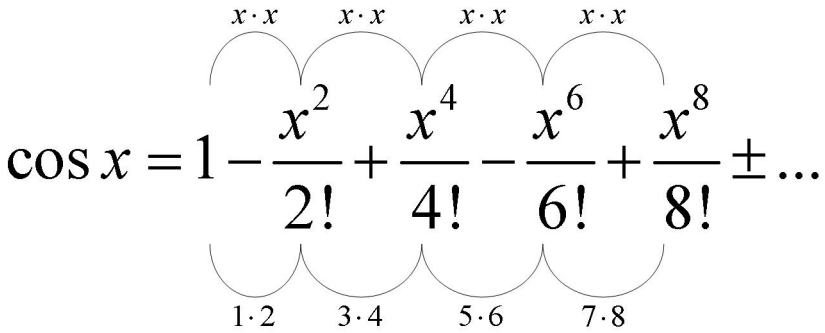
Methode zur Berechnung des Cosinus




```
public static double cos(double x)
{
  double zaehler = 1.0;
  double nenner = 1.0;
  double summe = 1.0;

  for (int i = 2;         ;     )
  {
    zaehler = zaehler * x * x * (-1);
    nenner = nenner * i * (i-1);
```

Der Nenner berechnet sich, indem man den alten Nenner mit dem Laufindex i und i-1 multipliziert.

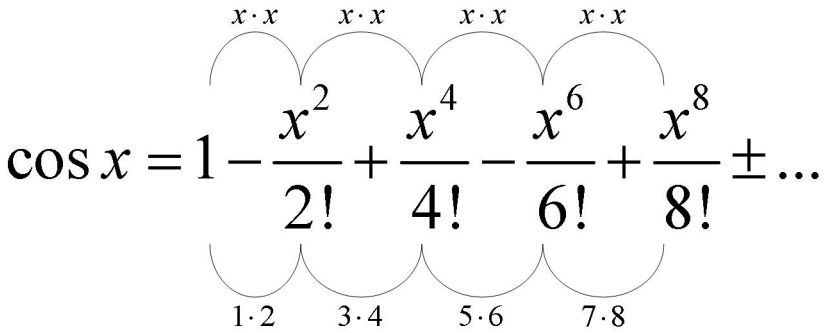
Methode zur Berechnung des Cosinus



Aufaddieren!

```
public static double cos(double x)
{
  double zaehler = 1.0;
  double nenner = 1.0;
  double summe = 1.0;
  double summand;
  for (int i = 2;         ;     )
  {
    zaehler = zaehler * x * x * (-1);
    nenner = nenner * i * (i-1);
    summand = zaehler/nenner;
    summe = summe + summand;
  }
```

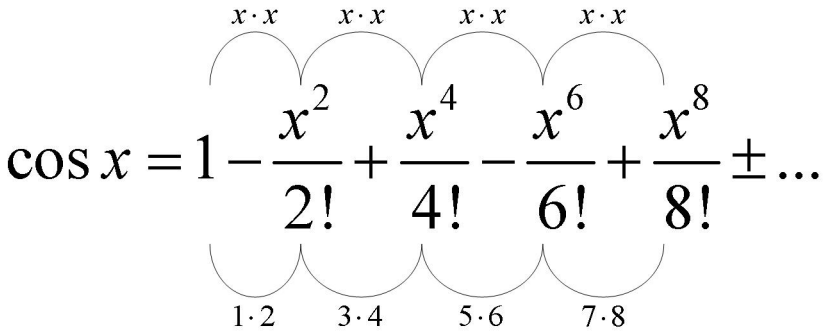
Methode zur Berechnung des Cosinus



Um wie viel muss i erhöht werden?

```
public static double cos(double x)
{
  double zaehler = 1.0;
  double nenner = 1.0;
  double summe = 1.0;
  double summand;
  for (int i = 2;         ;i=i+2)
  {
    zaehler = zaehler * x * x * (-1);
    nenner = nenner * i * (i-1);
    summand = zaehler/nenner;
    summe = summe + summand;
  }
```

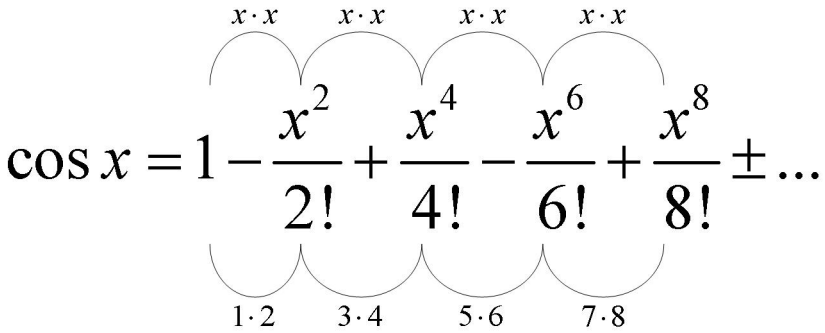
Methode zur Berechnung des Cosinus



Wie wählen wir die Laufbedingung?

```
public static double cos(double x)
{
  double zaehler = 1.0;
  double nenner = 1.0;
  double summe = 1.0;
  double summand;
  for (int i = 2;         ;i=i+2)
  {
    zaehler = zaehler * x * x * (-1);
    nenner = nenner * i * (i-1);
    summand = zaehler/nenner;
    summe = summe + summand;
  }
```

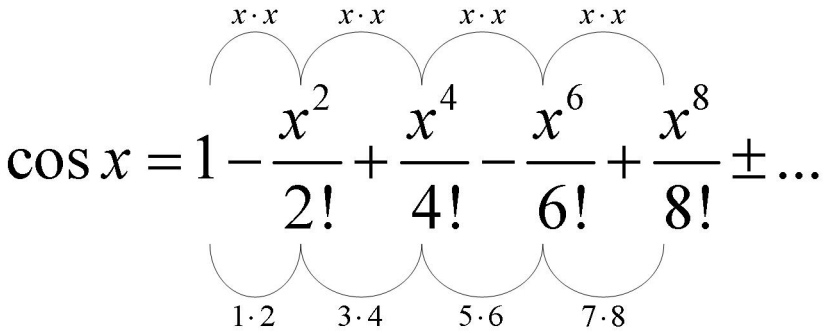
Methode zur Berechnung des Cosinus



Wie wählen wir die Laufbedingung?

```
public static double cos(double x)
{
  double zaehler = 1.0;
  double nenner = 1.0;
  double summe = 1.0;
  double summand = 1;
  for (int i = 2; summand > 1E-15 ||
                  summand < -1E-15; i=i+2)
  {
    zaehler = zaehler * x * x * (-1);
    nenner = nenner * i * (i-1);
    summand = zaehler/nenner;
    summe = summe + summand;
  }
```

Methode zur Berechnung des Cosinus



Rückgabewert ergänzen!

```
public static double cos(double x)
{
  double zaehler = 1.0;
  double nenner = 1.0;
  double summe = 1.0;
  double summand = 1;
  for (int i = 2; summand > 1E-15 ||
                  summand < -1E-15; i=i+2)
  {
    zaehler = zaehler * x * x * (-1);
    nenner = nenner * i * (i-1);
    summand = zaehler/nenner;
    summe = summe + summand;
  }
  return summe;
}
```

Methodendokumentation

```
/**
 * Berechnet den Kosinus von der übergebenen Zahl x.
 * @param x reelle Zahl (in RAD), von der der Kosinus berechnet werden soll.
 * @return Ergebnis des Kosinus; reelle Zahl zwischen -1 und 1 (beide inklusive)
 */
public static double cos(double x)
{
  double zaehler = 1.0;
  double nenner = 1.0;
  double summe = 1.0;
  double summand = 1;
  for (int i = 2; summand > 1E-15 || summand < -1E-15; i = i + 2)
  {
    zaehler = zaehler * x * x * (-1);
    nenner = nenner * i * (i - 1);
    summand = zaehler / nenner;
    summe = summe + summand;
  }
  return summe;
}
```

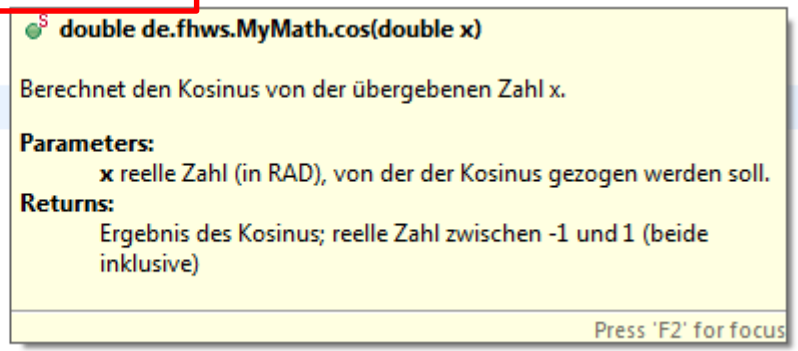
Wenn andere die Methode verwenden sollen, sollten wir sie entsprechend dokumentieren!

Anderen Methoden zur Verfügung stellen

```
package de.fhws;
public class MyMath {
  /**
   * Berechnet den Kosinus von der übergebenen Zahl x.
   * @param x reelle Zahl (in RAD), von der der Kosinus berechnet werden soll.
   * @return Ergebnis des Kosinus; reelle Zahl zwischen -1 und 1 (beide inklusive)
   */
  public static double cos(double x) {
    double zaehler = 1.0;
    double nenner = 1.0;
    double summe = 1.0;
    double summand = 1;
    for (int i = 2; summand > 1E-15 || summand < -1E-15; i = i + 2) {
      zaehler = zaehler * x * x * (-1);
      nenner = nenner * i * (i - 1);
      summand = zaehler / nenner;
      summe = summe + summand;
    }
    return summe;
  }
}
```

Methoden einbinden

```
package pkg06;

import de.fhws.MyMath;

public class KosinusTest
{
  public static void main(String[] args)
  {
    System.out.println(MyMath.cos(0.5));
  }
}
```



Um die Methode aufrufen zu können, muss die Klasse MyMath auf dem Klassenpfad liegen.

Mathematische Funktionen

- Die Klasse `java.lang.Math` stellt eine Menge an geläufigen mathematischen Konstanten und Funktionen zur Verfügung:

Mathematische Funktion oder Konstante	Methode oder Konstante in Java
e	double **Math.E**
π	double **Math.PI**
$\|x\|$	double **Math.abs**(double x)
$\cos(x)$	double **Math.cos**(double x)
$\sin(x)$	double **Math.sin**(double x)
$\sqrt{x}$	double **Math.sqrt**(double x)
x^y	double **Math.pow**(double x, double y)
gibt Zufallszahl x mit $x \in [0,1)$ zurück	double **Math.random**()

switch-Anweisung

- Eine switch-Anweisung ist eine Kontrollstruktur, die i.d.R. alternativ zu einer if-Anweisung mit mehreren else-ifs eingesetzt wird.
- Die switch-Anweisung nimmt
  - eine ganze Zahl (byte, char, short, int, aber **kein** long),
  - einen String
  - oder ein Enum (später dazu mehr)

  entgegen.

Wie sieht eine switch-Anweisung aus?

```
int zahl = (int) (Math.random() * 6 + 1);
switch (zahl)
{




}
```

```
int zahl = (int) (Math.random() * 6 + 1);
switch (zahl)
{
  case 1 -> System.out.println("Es wurde eine 1 gewürfelt.");




}
```

Das case label muss ein konstanter Ausdruck vom Typ der Variablen im switch sein

```java
int zahl = (int) (Math.random() * 6 + 1);
switch (zahl)
{
  case 1 -> System.out.println("Es wurde eine 1 gewürfelt.");
  case 2 -> System.out.println("Es wurde eine 2 gewürfelt.");
  case 3 -> System.out.println("Es wurde eine 3 gewürfelt.");
  case 4 -> System.out.println("Es wurde eine 4 gewürfelt.");
  case 5 -> System.out.println("Es wurde eine 5 gewürfelt.");
  case 6 -> System.out.println("Es wurde eine 6 gewürfelt.");
}
```

Es wird der `case` betreten, der mit **`zahl`** übereinstimmt.
Die Anweisung oder der Block hinter dem **->** wird ausgeführt.

```java
int zahl = (int) (Math.random() * 6 + 1);
switch (zahl)
{
  case 1 -> System.out.println("Es wurde eine 1 gewürfelt.");
  case 2 -> System.out.println("Es wurde eine 2 gewürfelt.");
  case 3 -> System.out.println("Es wurde eine 3 gewürfelt.");
  case 4 -> System.out.println("Es wurde eine 4 gewürfelt.");
  case 5 -> System.out.println("Es wurde eine 5 gewürfelt.");
  default -> System.out.println("Es wurde keine 1,2,3,4 oder 5 gewürfelt.");
}
```

Trifft kein `case` zu, wird die Anweisung hinter dem `default` case ausgeführt.

Dann wenden wir switch mal an!

Methode zur Zuordnung der Tagesanzahl für einen Monat

```
public static int tageImMonat(int monat)
{
  int tage;
  switch(monat)
  {
    case 2 -> tage = 28;
    case 1,3,5,7,8,10,12 -> tage = 31;
    case 4,6,9,11 -> tage = 30;
  }
  return tage;
}
```

The local variable **tage** may not have been initialized

Methode zur Zuordnung der Tagesanzahl für einen Monat

```
public static int tageImMonat(int monat)
{
  int tage = -1;
  switch(monat)
  {
    case 2 -> tage = 28;
    case 1,3,5,7,8,10,12 -> tage = 31;
    case 4,6,9,11 -> tage = 30;
  }
  return tage;
}
```

Methode zur Zuordnung der Tagesanzahl für einen Monat

```
public static int tageImMonat(int monat)
{
  int tage;
  switch(monat)
  {
    case 2 -> tage = 28;
    case 1,3,5,7,8,10,12 -> tage = 31;
    case 4,6,9,11 -> tage = 30;
    default -> tage = -1;
  }
  return tage;
}
```

Die Variable **tage** wird in jedem Fall belegt, im Zweifelsfall durch den **default case**.

switch kann auch als **Ausdruck** verwendet werden.

Methode zur Zuordnung von Monaten zu Jahreszeiten

Switch-Anweisung

```
public static int tageImMonat(int monat)
{
  int tage;
  switch(monat)
  {
    case 2 -> tage = 28;
    case 1,3,5,7,8,10,12 -> tage = 31;
    case 4,6,9,11 -> tage = 30;
    default -> tage = -1;
  }
  return tage;
}
```

Switch-Ausdruck

```
public static int tageImMonat (int monat)
{

  int tage = switch(monat)
  {
    case 2 -> 28;
    case 1,3,5,7,8,10,12 -> 31;
    case 4,6,9,11 -> 30;
    default -> -1;
  };
  return tage;
}
```

Hinter dem **->** von jedem Case steht keine Anweisung, sondern ein Ausdruck.

Ein switch-Ausdruck muss für jede Variablenbelegung (von monat) evaluiert werden können.

Neben **int** lassen sich auch **Strings** in einem switch verwenden (wie auch byte, short und Enums).

Methode zur Zuordnung der Tagesanzahl für einen Monat

```
public static int tageImMonat(String monat)
 {
   int tage = switch(monat)
   {
     case "Februar" -> 28;
     case "April", "Juni", "September", "November" -> 30;
     case "Januar", "März", "Mai", "Juli", "August", "Oktober", "Dezember" -> 31;
     default -> -1;
   };
   return tage;
 }
```

Wenn mehrere Anweisungen in einem case benötigt werden,
kann ein Block verwendet werden.

Switch-Ausdruck

Switch-Anweisung

```
public static int tageImMonat(int monat)
{
  int tage;
  switch(monat)
  {
    case 2 -> tage = 28;
    case 1,3,5,7,8,10,12 -> tage = 31;
    case 4,6,9,11 -> tage = 30;
    default ->
    {
      System.out.println("Falscher Monatsname");
      tage = -1;
    }
  }
  return tage;
}
```

Wenn mehr als eine Anweisung hinter einem case label benötigt wird, können wir einen Block verwenden.

Switch-Ausdruck

```
public static int tageImMonat (int monat)
{
  int tage = switch(monat)
  {
    case 2 -> 28;
    case 1,3,5,7,8,10,12 -> 31;
    case 4,6,9,11 -> 30;
    default ->
    {
      System.out.println("Falscher Monatsname");
      yield -1;
    }
  };
  return tage;
}
```

Wenn ein Block in einem Switch-**Ausdruck** verwendet wird, wird die Anweisung yield benötigt, um das Ergebnis zu speichern.

Nehmen wir an, wir wollen 10000 Würfelwürfe simulieren!

```
int einser = 0, zweier = 0, dreier = 0,
    vierer = 0, fuenfer = 0, sechser = 0;

for (int i = 0; i < 10000; i++)
{
  int wurf = (int)(Math.random()*6+1);
  switch(wurf) {
    case 1 -> einser++;
    case 2 -> zweier++;
    case 3 -> dreier++;
    case 4 -> vierer++;
    case 5 -> fuenfer++;
    case 6 -> sechser++;
  }
}
System.out.println("1er: " + einser);
System.out.println("2er: " + zweier);
System.out.println("3er: " + dreier);
System.out.println("4er: " + vierer);
System.out.println("5er: " + fuenfer);
System.out.println("6er: " + sechser);
```

```
int einser = 0, zweier = 0, dreier = 0,
    vierer = 0, fuenfer = 0, sechser = 0;

for (int i = 0; i < 10000; i++)
{
  int wurf = (int)(Math.random()*6+1);
  switch(wurf) {
    case 1 -> einser++;
    case 2 -> zweier++;
    case 3 -> dreier++;
    case 4 -> vierer++;
    case 5 -> fuenfer++;
    case 6 -> sechser++;
  }
}
System.out.println("1er: " + einser);
System.out.println("2er: " + zweier);
System.out.println("3er: " + dreier);
System.out.println("4er: " + vierer);
System.out.println("5er: " + fuenfer);
System.out.println("6er: " + sechser);
```

gleichartige Informationen:
Zähle bei einer 1 die Einser hoch
Zähle bei einer 2 die Zweier hoch
etc.

Kann man die verschiedenen Variablen kurz auf gleiche Weise behandeln?

Arrays

- Arrays (auch Felder genannt) werden verwendet, um mehrere Variablen des gleichen Datentyps zu gruppieren.

```
/*Deklariere eine Variable, die eine Referenz auf ein Feld von Integervariablen
   speichert*/
int[] zahlen;

/*Lege Speicher für ein Feld von 6 Integervariablen an
  Der new-Operator gibt eine Referenz auf den angelegten Speicherbereich zurück. */
zahlen = new int[6];


/* Beides in einer Zeile:
   Deklariere (ein Feld mit) 6 Integervariablen*/
int[] zahlen = new int[6];
```

Arrays

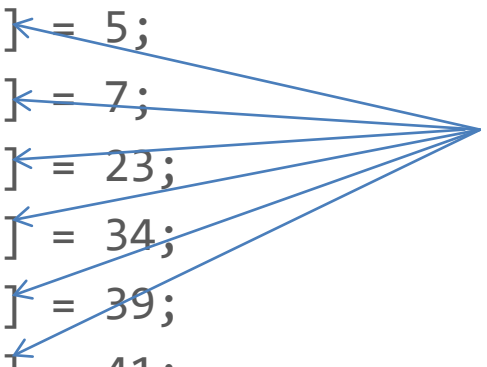
- Nehmen wir an, wir wollen die Lottozahlen speichern:

```
/*Deklariere (ein Feld mit) 6 Integervariablen*/
int[] zahlen = new int[6];

//In den folgenden Zeilen werden diese mit Werten belegt:
zahlen[0] = 5;
zahlen[1] = 7;
zahlen[2] = 23;
zahlen[3] = 34;
zahlen[4] = 39;
zahlen[5] = 41;
```

Arrays

- Nehmen wir an, wir wollen die Lottozahlen speichern:

```
/*Deklariere (ein Feld mit) 6 Integervariablen*/
int[] zahlen = new int[6];

//In den folgenden Zeilen werden diese mit Werten belegt:
zahlen[0] = 5;
zahlen[1] = 7;
zahlen[2] = 23;
zahlen[3] = 34;
zahlen[4] = 39;
zahlen[5] = 41;
```

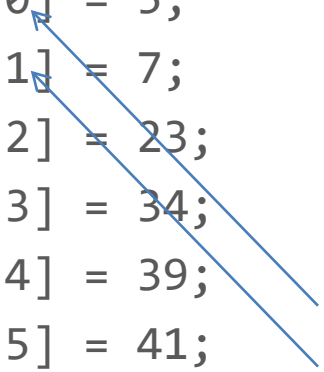


Der Index in den eckigen Klammern gibt an, auf welche Variable des Arrays zugegriffen wird.

Arrays

- Nehmen wir an, wir wollen die Lottozahlen speichern:

```
/*Deklariere (ein Feld mit) 6 Integervariablen*/
int[] zahlen = new int[6];

//In den folgenden Zeilen werden diese mit Werten belegt:
zahlen[0] = 5;
zahlen[1] = 7;
zahlen[2] = 23;
zahlen[3] = 34;
zahlen[4] = 39;
zahlen[5] = 41;
```



Das erste Element des Arrays ist über den Index „0“ zugreifbar.
Das zweite Element über den Index „1“, etc.

Initialisierung und Länge von Arrays

- Eine direkte Initialisierung des Arrays kann in der Deklarationszeile erfolgen:

```
int[] zahlen = new int[]{5, 7, 23, 34, 39, 41};
```

oder kürzer

```
int[] zahlen = {5, 7, 23, 34, 39, 41};
```

- Von einem Array kann man über dessen Attribut ***length*** die Anzahl der Elemente im Array abfragen.

- *zahlen.length* ergibt in obigem Bsp. den Wert *6*.

```
int einser = 0, zweier = 0, dreier = 0,
    vierer = 0, fuenfer = 0, sechser = 0;

for (int i = 0; i < 10000; i++)
{
  int wurf = (int)(Math.random()*6+1);
  switch(wurf) {
    case 1 -> einser++;
    case 2 -> zweier++;
    case 3 -> dreier++;
    case 4 -> vierer++;
    case 5 -> fuenfer++;
    case 6 -> sechser++;
  }
}
System.out.println("1er: " + einser);
System.out.println("2er: " + zweier);
System.out.println("3er: " + dreier);
System.out.println("4er: " + vierer);
System.out.println("5er: " + fuenfer);
System.out.println("6er: " + sechser);
```

```
int[] gewuerfelteSeiten = new int[6];


for (int i = 0; i < 10000; i++)
{
  int wurf = (int)(Math.random()*6+1);
  switch(wurf) {
    case 1 -> einser++;
    case 2 -> zweier++;
    case 3 -> dreier++;
    case 4 -> vierer++;
    case 5 -> fuenfer++;
    case 6 -> sechser++;
  }
}
System.out.println("1er: " + einser);
System.out.println("2er: " + zweier);
System.out.println("3er: " + dreier);
System.out.println("4er: " + vierer);
System.out.println("5er: " + fuenfer);
System.out.println("6er: " + sechser);
```

```
int[] gewuerfelteSeiten = new int[6];


for (int i = 0; i < 10000; i++)
{
  int wurf = (int)(Math.random()*6+1);
  switch(wurf) {
    case 1 -> gewuerfelteSeite[0]++;
    case 2 -> gewuerfelteSeite[1]++;
    case 3 -> gewuerfelteSeite[2]++;
    case 4 -> gewuerfelteSeite[3]++;
    case 5 -> gewuerfelteSeite[4]++;
    case 6 -> gewuerfelteSeite[5]++;
  }
}
System.out.println("1er: " + einser);
System.out.println("2er: " + zweier);
System.out.println("3er: " + dreier);
System.out.println("4er: " + vierer);
System.out.println("5er: " + fuenfer);
System.out.println("6er: " + sechser);
```

```
int[] gewuerfelteSeiten = new int[6];


for (int i = 0; i < 10000; i++)
{
  int wurf = (int)(Math.random()*6+1);
  switch(wurf) {
    case 1 -> gewuerfelteSeite[0]++;
    case 2 -> gewuerfelteSeite[1]++;
    case 3 -> gewuerfelteSeite[2]++;
    case 4 -> gewuerfelteSeite[3]++;
    case 5 -> gewuerfelteSeite[4]++;
    case 6 -> gewuerfelteSeite[5]++;
  }
}
for (int i = 0; i < gewuerfelteSeiten.length; i++)
{
  System.out.println(i+1+"er: " + gewuerfelteSeiten[i]);
}
```

```
int[] gewuerfelteSeiten = new int[6];


for (int i = 0; i < 10000; i++)
{
  int wurf = (int)(Math.random()*6+1);
  switch(wurf) {
    case 1 -> gewuerfelteSeite[wurf-1]++;
    case 2 -> gewuerfelteSeite[wurf-1]++;
    case 3 -> gewuerfelteSeite[wurf-1]++;
    case 4 -> gewuerfelteSeite[wurf-1]++;
    case 5 -> gewuerfelteSeite[wurf-1]++;
    case 6 -> gewuerfelteSeite[wurf-1]++;
  }
}
for (int i = 0; i < gewuerfelteSeiten.length; i++)
{
  System.out.println(i+1+"er: " + gewuerfelteSeiten[i]);
}
```

Die Anweisung in jedem Case ist jetzt gleich.

Die Switch-Anweisung wird jetzt nicht mehr benötigt.

```
int[] gewuerfelteSeiten = new int[6];


for (int i = 0; i < 10000; i++)
{
  int wurf = (int)(Math.random()*6+1);




  gewuerfelteSeiten[wurf-1]++;



}
for (int i = 0; i < gewuerfelteSeiten.length; i++)
{
  System.out.println(i+1+"er: " + gewuerfelteSeiten[i]);
}
```

Wir wollen ein Feld von Zahlen sortieren, so dass die kleinste zuerst kommt.

Sortieren

68	22	56	34	12	38	9	17

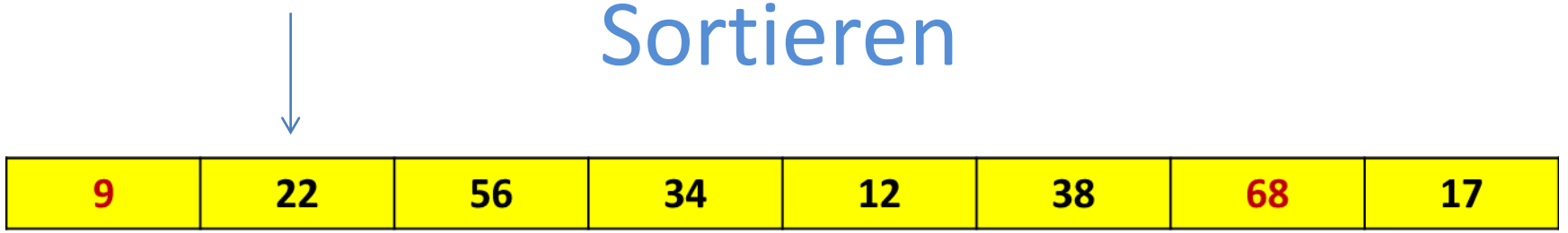
- Der Inhalt der ersten Feld-Komponente wird als Minimum angesehen (68)!
- In den rechts davon liegenden Feldkomponenten wird ein neues Minimum gesucht:

```
➢ 68 > 22 →   22 neues Minimum
➢ 22 < 56 → kein neues Minimum
➢ 22 < 34 → kein neues Minimum
➢ 22 > 12 →   12 neues Minimum
➢ 12 < 38 → kein neues Minimum
➢ 12 >  9 →    9 neues Minimum
➢  9 < 17 → kein neues Minimum
```

68 wird mit 9 getauscht !

9	22	56	34	12	38	68	17

Sortieren

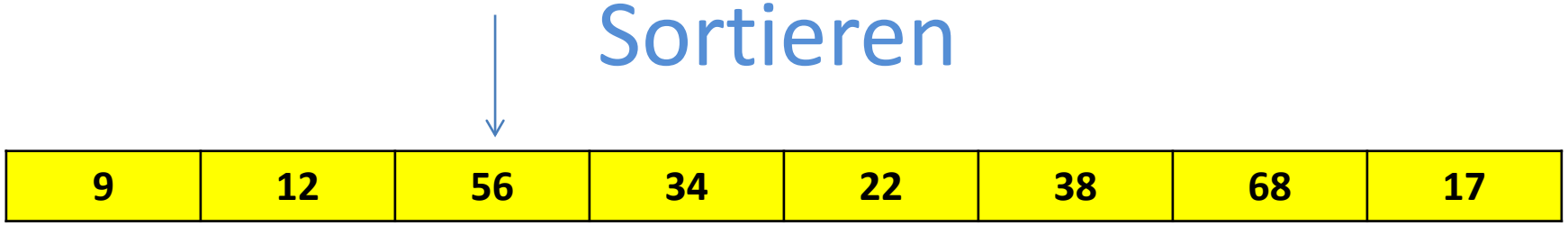


- Der Inhalt der zweiten Feld-Komponente wird als Minimum angesehen (22)!
- In den rechts davon liegenden Feldkomponenten wird ein neues Minimum gesucht:

➢ 22 < 56 → kein neues Minimum
➢ 22 < 34 → kein neues Minimum
➢ 22 > 12 → 12 neues Minimum
➢ 12 < 38 → kein neues Minimum
➢ 12 < 68 → kein neues Minimum
➢ 12 < 17 → kein neues Minimum

22 wird mit 12 getauscht !

Sortieren



- Der Inhalt der dritten Feld-Komponente wird als Minimum angesehen (56)!
- In den rechts davon liegenden Feldkomponenten wird ein neues Minimum gesucht:

```
➢ 56 > 34 →   34 neues Minimum
➢ 34 > 22 →   22 neues Minimum
➢ 22 < 38 → kein neues Minimum
➢ 22 < 68 → kein neues Minimum
➢ 22 > 17 →   17 neues Minimum
```

56 wird mit 17 getauscht !

usw.

Finch

source: http://www.finchrobot.com/sites/default/files/BirdBrain1010-0005.jpg

- Lichtsensor
- Temperatursensor
- Sensor für Hindernisse
- Beschleunigungsmesser
- Motoren
- Buzzer
- RGB “Schnabel”-LED
- Stiftbefestigung (zum Malen)
- Strom über USB Kabel

source: http://www.finchrobot.com/sites/default/files/finchie.jpg

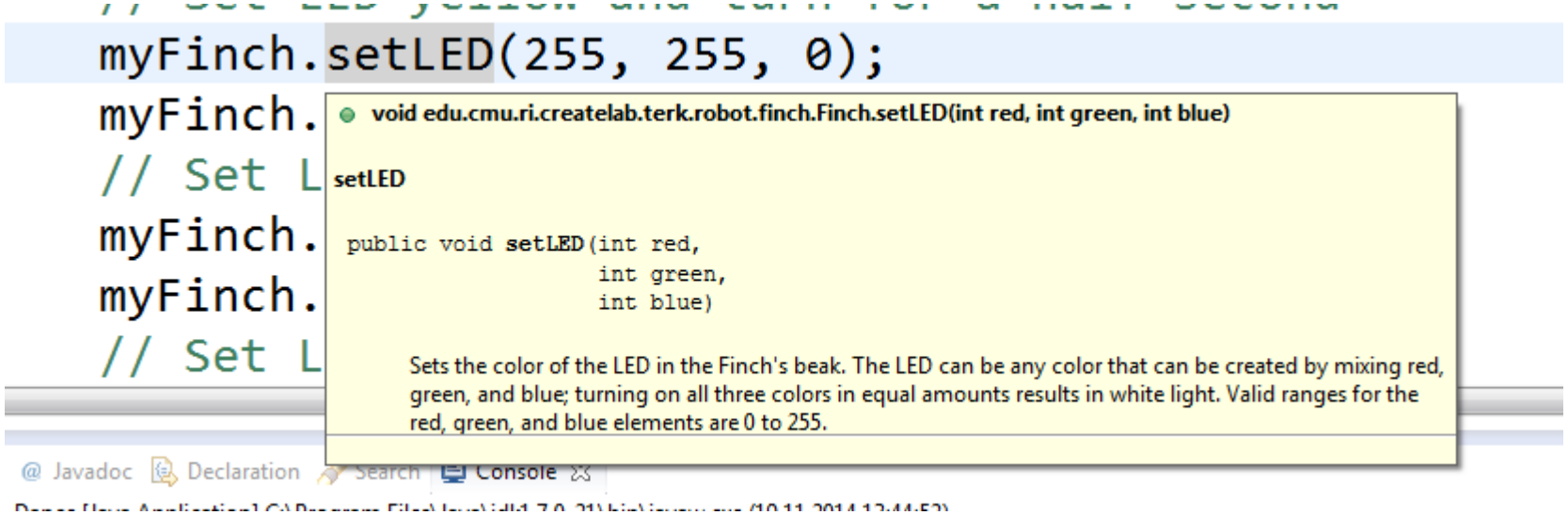
Finch einrichten

- von https://www.birdbraintechnologies.com/ Eclipse package herunterladen!
- FinchEclipseJavaXXX.zip entpacken
- Ordner FinchEclipse in Workspace-Ordner kopieren
- (den aktuell verwendeten Workspace, findet man unter File-Switch Workspace)
- Ordner FinchEclipse in Eclipse importieren:
  - File -> Import
  - General
  - Existing Projects into Workspace

- Stellen Sie sicher, dass Sie ein Oracle JDK (anstelle von OpenJDK) verwenden!

Javadoc einrichten

- Um die Hilfe zu den Methoden zu sehen,



muss das JavaDoc eingerichtet werden.

Javadoc einrichten

- Rechte Taste auf das Projekt -> Properties
- Java Build Path -> Libraries -> finch.jar aufklappen
- *Javadoc location* anklicken und Edit auswählen
- Unter JavaDoc URL den Pfad des javadoc Ordners auswählen:

  file:/<absoluter Pfad zum Eclipse Workspace>/<projektname>/javadoc/
- z.B.
  - file:/C:/W/WorkspaceProgrammierenI/Finch2/javadoc/

Finch testen

- Finch per USB an den Rechner anschließen
- Dance.java in Eclipse starten