

Lektion 3

Eingabe, Bedingte Anweisungen, Verzweigungen

Eingabe

Mittlerweile können wir

- Berechnungen anstellen
- Daten ausgeben

d.h. wir können vom EVA-Prinzip die **Verarbeitung**
und **Ausgabe** bereits umsetzen

Jetzt schauen wir uns an, wie man Daten einliest!

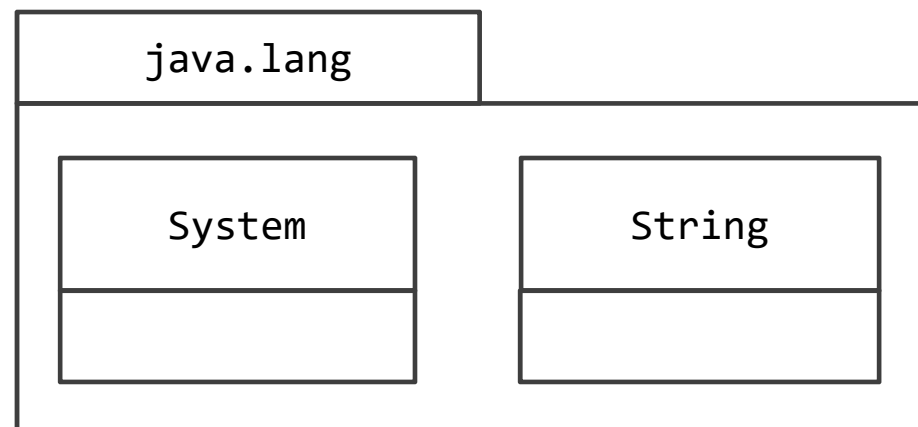
Eingabe und Ausgabe

- Ein- und Ausgabe (Input/Output; I/O) erfolgen durch Streams
- Streams lesen
 - von der Tastatur (Standardinput)
 - aus einer Datei
 - aus dem Hauptspeicher
 - von Sockets (Netzwerk)
- Streams schreiben
 - auf den Standardoutput (wird i.d.R. weitergeleitet auf Konsole/Terminal)
 - in eine Datei
 - in den Hauptspeicher
 - auf Sockets (Netzwerk)

Bisher haben wir neben unserer Hauptklasse
(die, in der die main-Methode definiert war)
folgende Klassen verwendet:

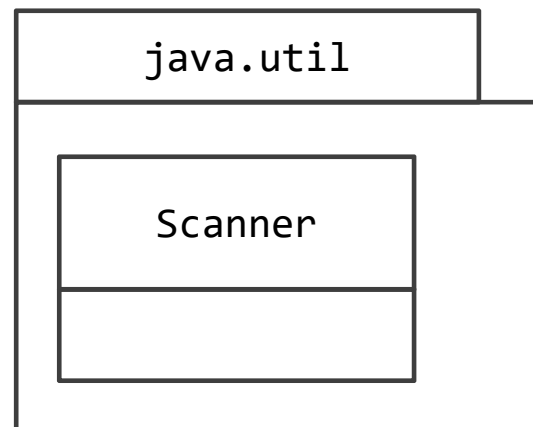
System
String

Beide Klassen sind im Package `java.lang` definiert.



Klassen aus dem Package *java.Lang* können wir einfach so in unseren Programmen verwenden.

Das Einlesen von der Tastatur erfolgt über die Verwendung der Klasse *Scanner* des Packages *java.util*



Klassen, die nicht im eigenen Package oder im Package *java.Lang* liegen, können über **import** eingebunden werden.

Bindet die Klasse Scanner in unsere Klasse ein und ermöglicht deren Verwendung.

```
import java.util.Scanner;

public class Einlesen
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        ...
    }
}
```

Objekt, das in Java die Standardeingabe (die Eingabe vom Keyboard) repräsentiert

Von der Tastatur lesen

Was passiert bei folgendem Code?

```
Scanner scanner = new Scanner(System.in);
```

`new Scanner(..)` belegt Speicherplatz für ein neues **Objekt** der **Klasse** Scanner
(Klassen dienen als Baupläne/Schablonen)

und erstellt dieses.

In der Objektreferenz `scanner` wird ein Verweis auf das Scanner-Objekt gespeichert.

Funktionsweise Scanner

- Tastatureingabe mit `java.util.Scanner`

2

5

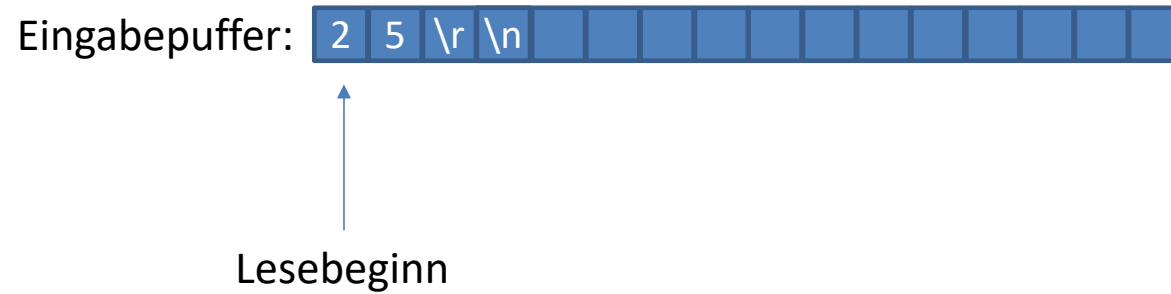
\r\n

Eingabepuffer:

| | | | | | | | | | | | | | | | | | | |
|---|---|----|----|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| 2 | 5 | \r | \n | | | | | | | | | | | | | | | |
|---|---|----|----|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|

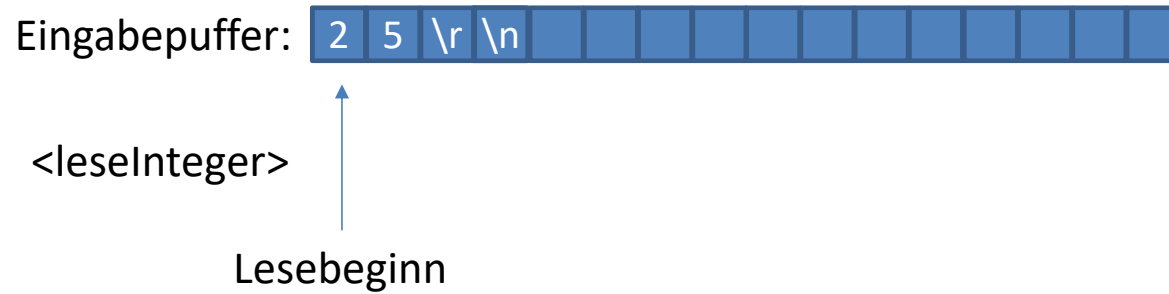
Funktionsweise Scannern

- Lese Integer mit `scanner.nextInt()`



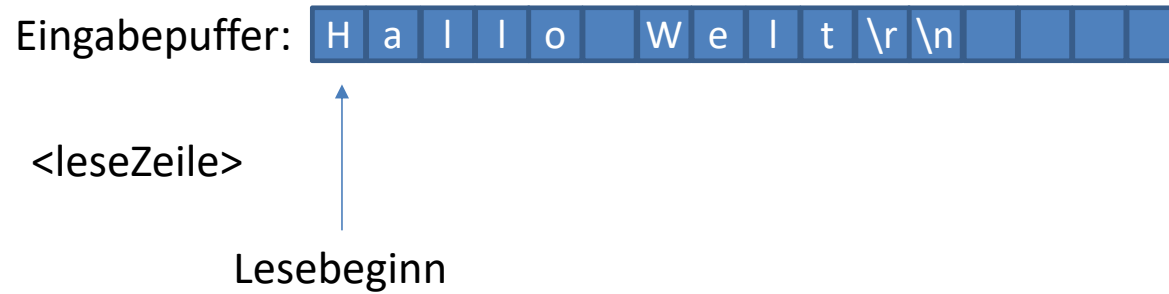
Funktionsweise Scanner

- Lese Integer mit `scanner.nextInt()`



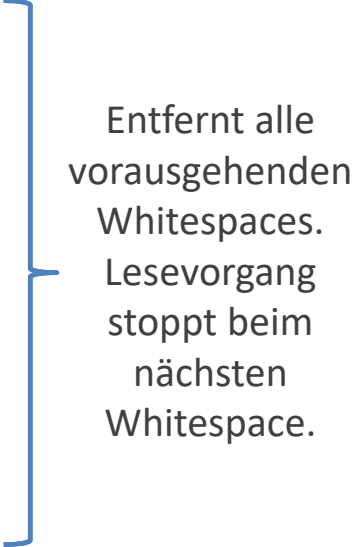
Funktionsweise Scanner

- Lese String mit `scanner.nextLine()`



Scanner

- Bei folgenden Methodenaufrufen wartet das Programm blockierend bis Daten aus dem Eingabepuffer zum Lesen zur Verfügung stehen.

| Scanner-Methode | Funktionsweise | |
|----------------------------|---|--|
| <code>nextInt()</code> | Liest den nächsten Integer aus dem Eingabepuffer. |  <p>Entfernt alle vorausgehenden Whitespaces. Lesevorgang stoppt beim nächsten Whitespace.</p> |
| <code>nextByte()</code> | Liest das nächste Byte aus dem Eingabepuffer. | |
| <code>nextShort()</code> | Liest den nächsten Short aus dem Eingabepuffer. | |
| <code>nextLong()</code> | Liest den nächsten Long aus dem Eingabepuffer. | |
| <code>nextFloat()</code> | Liest den nächsten Float aus dem Eingabepuffer. | |
| <code>nextDouble()</code> | Liest den nächsten Double aus dem Eingabepuffer. | |
| <code>nextBoolean()</code> | Liest den nächsten Bool'schen Wert aus dem Eingabepuffer. (Später mehr dazu!) | |
| <code>nextLine()</code> | Liest die nächste Zeile (bis zum nächsten Zeilenumbruch) als String ein. | |

Eingabe/Input

- Anstatt Variablenwerte im Quellcode festzusetzen (z. B. `x = 5`), können diese auch vom Benutzer eingelesen werden.

```
import java.util.Scanner; //fremde Klasse ins Programm einbinden

public class ScannerBeispiel
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Bitte geben Sie eine Zahl ein: ");
        int x = scanner.nextInt(); //int von Standardeingabe lesen
        System.out.print("Bitte geben Sie noch eine Zahl ein: ");
        int y = scanner.nextInt();
        System.out.print("Die Summe der beiden Zahlen ist: ");
        int result = x + y; // Summanden addieren und in Ergebnis speichern
        System.out.println(result);
    }
}
```

Kommazahlen einlesen – Beispiel

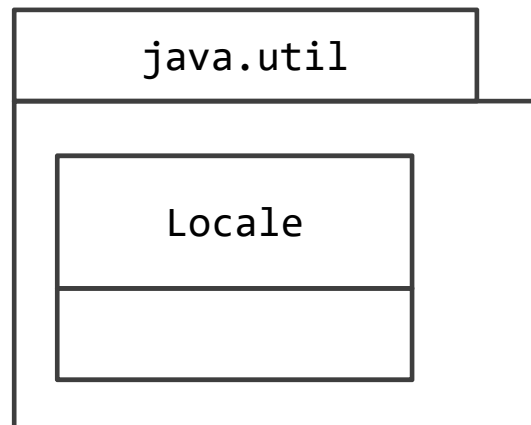
```
float kapital; //Kommazahl deklarieren
float zinssatz = 3.5f;
float kapitalNachEinemJahr;
System.out.println("Bitte geben Sie Ihr Grundkapital ein: ");
Scanner scanner = new Scanner(System.in);
kapital = scanner.nextFloat();
kapitalNachEinemJahr = kapital * (1.0f+(zinssatz/100.0f));
System.out.println("Verzinstes Kapital:" + kapitalNachEinemJahr);
```

Kommazahlen einlesen – Beispiel

Bitte beachten:

- Vom Standardinput gelesene Kommazahlen werden je nach Umgebung (Sprache/Betriebssystem)
 - mit Punkt (z. B. 1.0) (im Englischen) eingeben
 - mit Komma (z. B. 1,0) (im Deutschen) eingeben

Über die Klasse `Locale` lassen sich Sprache und Region des Java-Programms einstellen.



Kommazahlen einlesen - Locale

- Wenn die Eingabe der Kommazahlen mit Punkt erfolgen soll, kann die Umgebung der JVM durch Setzen eines **Locale** bspw. auf Englisch umgestellt werden:

```
import java.util.Locale;  
import java.util.Scanner;
```

Sprache
(z. B. en, de, ...)

Region
(z. B. US, DE, ...)

```
public class KommazahleneingabeMitLocale {  
    public static void main(String[] args) {  
        Locale.setDefault(new Locale("en", "US"));  
        Scanner s = new Scanner(System.in);  
        double d = s.nextDouble();  
        ...  
    }  
}
```

Eingabe erfolgt jetzt
– wie im Englischen –
mit Punkt.
Beispiel: 3.4 statt 3,4

String/Zeichenkette einlesen und ausgeben

```
System.out.print("Bitte geben Sie Ihren Namen ein: ");  
Scanner scanner = new Scanner(System.in);  
String name = scanner.nextLine();  
System.out.println("Sie haben folgenden Namen eingegeben: " + name);
```

Von der Tastatur lesen - Einzelzeichen

- Direkt über **System.in**

Fehler einfach an den Standardfehlerkanal weiterleiten; Programm wird beendet.
Auf diese Weise dürfen später **NIE** Fehler behandelt werden

```
public static void main(String[] args) throws IOException
{
    char c = (char) System.in.read();
    System.out.println(c);
}
```

- liest das nächste Zeichen aus dem Eingabepuffer; wartet blockierend bei leerem Puffer auf die nächste Eingabe

Von der Tastatur lesen - Einzelzeichen

- Oder über `scanner.nextLine()`;

```
Scanner scanner = new Scanner(System.in);  
String line = scanner.nextLine();  
char c = line.charAt(0);  
System.out.println(c);
```

- die Methode `charAt(i)` eines String-Objekts gibt den Buchstaben an der i-ten Stelle als `char` zurück.
- Bsp.:
 - `"Welt".charAt(0)` gibt `'W'` zurück
 - `"Welt".charAt(1)` gibt `'e'` zurück

Scanner schließen

- Um belegte Ressourcen freizugeben, sollte der Scanner ordnungsgemäß geschlossen werden:

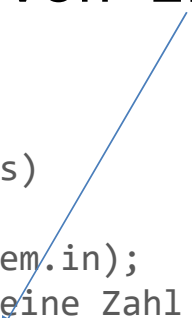
```
Scanner scanner = new Scanner(System.in);  
String line = scanner.nextLine();  
char c = line.charAt(0);  
System.out.println(c);  
scanner.close();
```

Vorsicht: `System.in` ist ein Stream.
`scanner.close()` schließt den
darunterliegenden Stream.

- Einen Scanner, der von `System.in` liest,
ausschließlich am Programmende schließen!

Was passiert bei folgendem Code
bei der Eingabe von 25?

```
public static void main(String[] args)
{
    Scanner scanner = new Scanner(System.in);
    System.out.print("Bitte geben Sie eine Zahl ein: ");
    System.out.println(scanner.nextInt());
    System.out.print("Bitte geben Sie einen String ein: ");
    System.out.println(scanner.nextLine());
}
```



Bedingte Anweisungen

if

Bisher haben unsere Programme einen festen Ablauf.

Manchmal ist es sinnvoll, Anweisungen an eine Bedingung zu knüpfen und ansonsten zu überspringen.

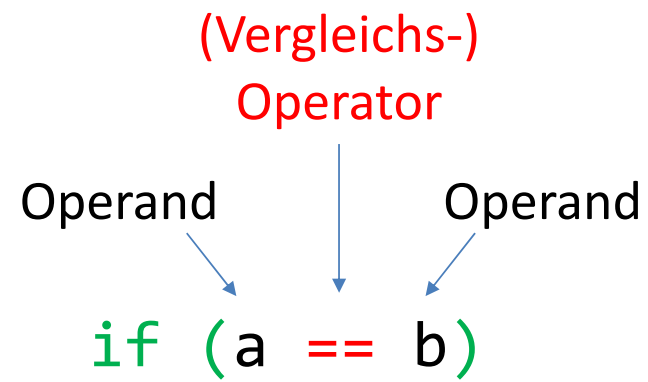
```
double temperatur = ...;
...
if (temperatur < 20) {
    //schalte Heizung ein
}
```

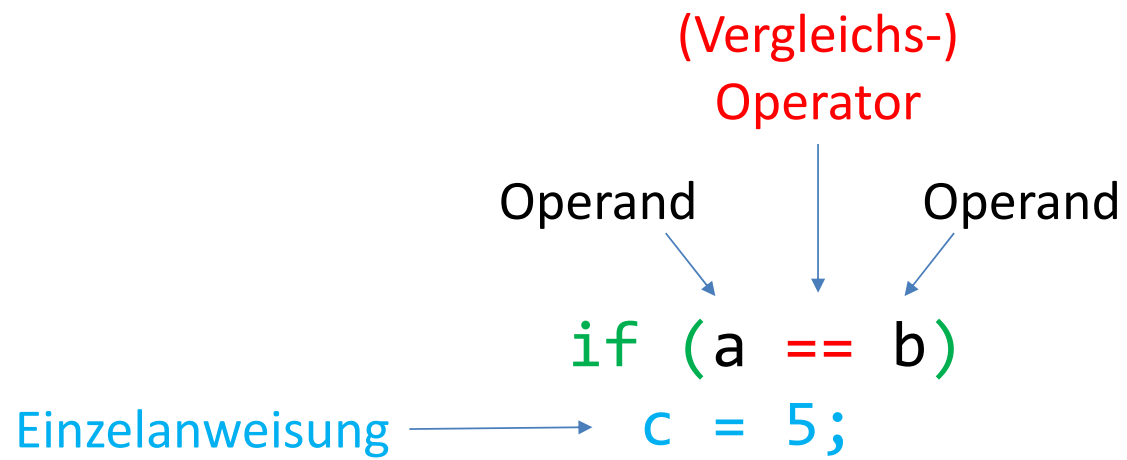
Bedingte Anweisung

(Vergleichs-)
Operator

Operand Operand

if (a == b)





(Vergleichs-)
Operator

Operand Operand

if (a == b) c = 5;

Einzelanweisung

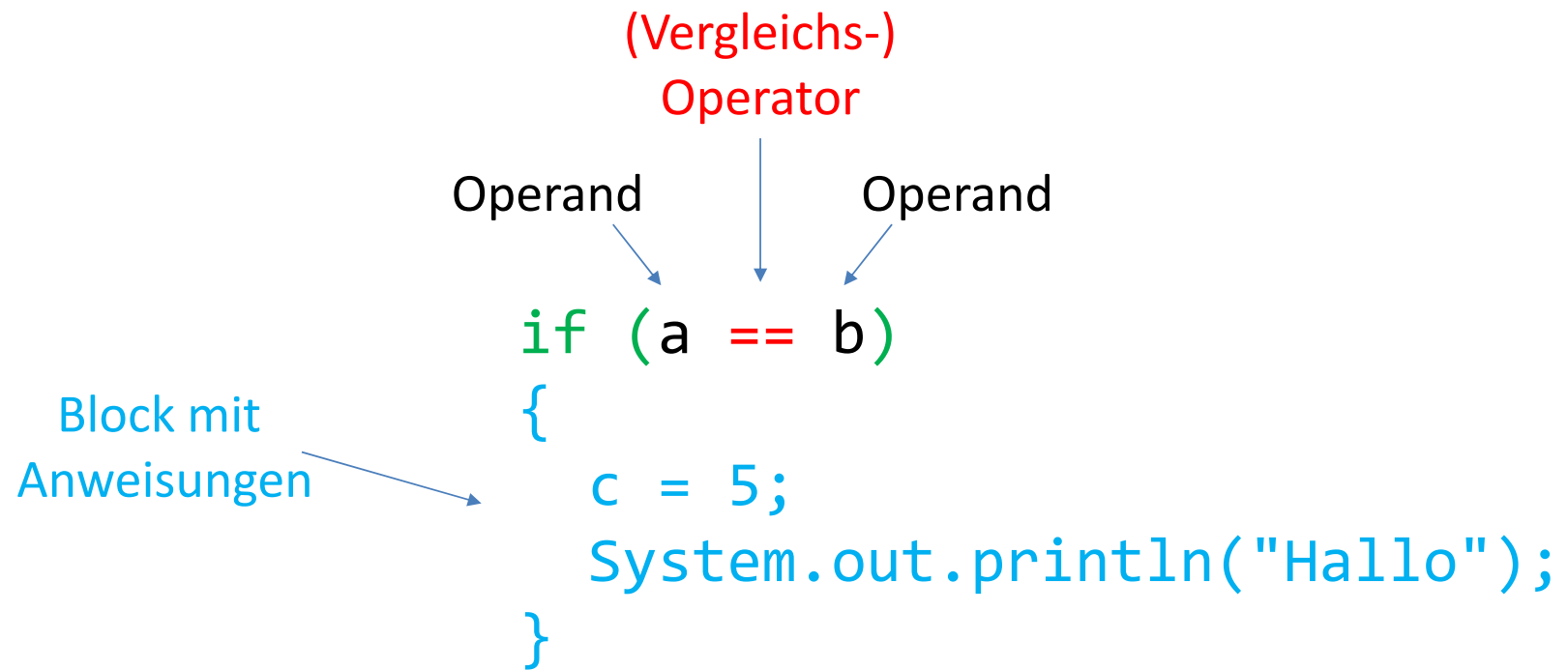
```
graph TD; A["(Vergleichs-) Operator"] --> B["a == b"]; C["Operand"] --> D["a"]; E["Operand"] --> F["b"]; G["Einzelanweisung"] --> H["c = 5;"]
```

(Vergleichs-)
Operator

Operand Operand

if (a == b); leere Anweisung

The diagram illustrates the components of the C statement `if (a == b);`. Above the code, the text `(Vergleichs-) Operator` in red has a blue arrow pointing to the `==` operator. Two instances of the word `Operand` in black have blue arrows pointing to the `a` and `b` operands respectively. To the right, the text `leere Anweisung` in blue has a blue arrow pointing to the semicolon `;` at the end of the statement.



Was ist eigentlich ein Block?

```
{  
    int b = 7;  
    System.out.println(b);  
}
```

Ein **Block** beginnt mit einer geschweiften Klammer...

...beinhaltet eine Folge von Anweisungen...

...und endet mit einer geschweiften Klammer.

Ein **Block** kann an den gleichen Stellen wie eine Einzelanweisung verwendet werden.

```
public static void main(String[] args)
{
    int a = 5;
    System.out.println(a);

    {
        int b = 7;

        System.out.println(b);
    }
}
```


Aber: Variablen sind nur innerhalb ihres Blocks sichtbar.

```
public static void main(String[] args)
{
    int a = 5;
    System.out.println(a);

    {
        int b = 7;
        System.out.println(a); //ok
        System.out.println(b);
    }
System.out.println(b); //b nicht sichtbar
}
```

Aber: Variablen sind nur innerhalb ihres Blocks sichtbar.

```
public static void main(String[] args)
{
    int a = 5;
    System.out.println(a);

    {
        int b = 7;
        System.out.println(a); //ok          b ist sichtbar
        System.out.println(b);
    }
    System.out.println(b); //b nicht sichtbar
}
```

Aber: Variablen sind nur innerhalb ihres Blocks sichtbar.

```
public static void main(String[] args)
{
    int a = 5;
    System.out.println(a);           a ist sichtbar
    {
        int b = 7;
        System.out.println(a); //ok   b ist sichtbar
        System.out.println(b);
    }
    System.out.println(b); //b nicht sichtbar
}
```

Blockbeispiel

```
public static void main(String[] args)
{
    {
        int x = 5;
        int z = 7;
        System.out.println(x+z);
    }

    {
        int x = 10;
        int y = 5;
        System.out.println(x-y);
    }
}
```

Blockbeispiel

```
public static void main(String[] args)
{
    {
        int x = 5;
        int z = 7;
        System.out.println(x+z);
    }

    {
        int x = 10;
        int y = 5;
        System.out.println(x-y);
        System.out.println(z); //z nicht sichtbar
    }
}
```

Verzweigungen

if und else

Wir haben gelernt, wie man mit if-Anweisungen Anweisungen an Bedingungen knüpft und sie ansonsten überspringt.

Unterschiedliche Benutzereingaben müssen ggf. unterschiedlich behandelt werden.

```
Changing TCP/IP Settings

1. Change Settings to DHCP
2. Change Settings to STATIC
3. Exit Program

Please make a selection [1-3]: _
```

source: <http://stackoverflow.com/questions/22773684/creating-a-console-menu-with-box-characters>

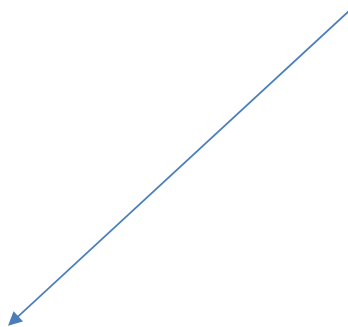
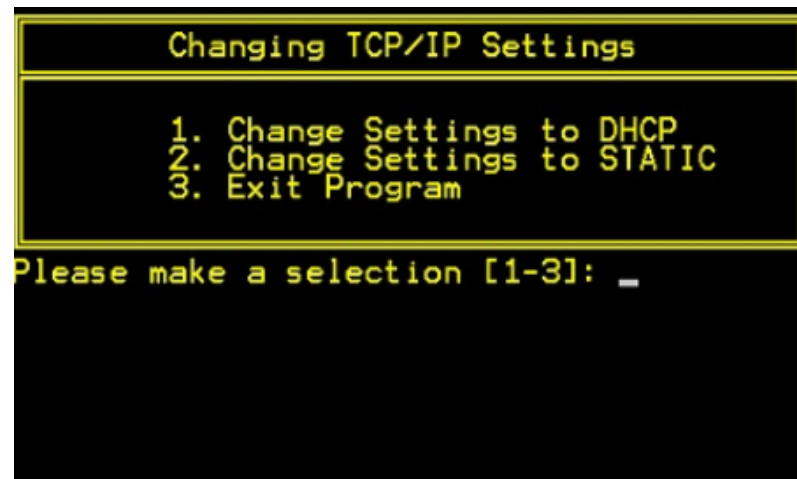
Changing TCP/IP Settings

1. Change Settings to DHCP

2. Change Settings to STATIC

3. Exit Program

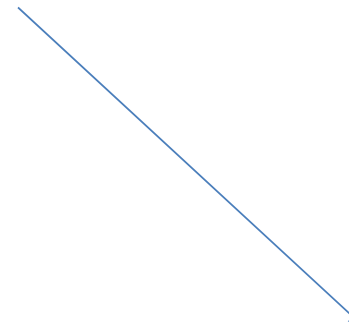
Please make a selection [1-3]: _



Change Settings to DHCP



Change Settings to STATIC



Exit Program

Daher sind **Verzweigungen** in einem Programm sinnvoll.

Verzweigungen

- Durch Verzweigungen ist es möglich
 - verschiedene Aufgaben aus einem Menü **auszuwählen**,
 - **Fallunterscheidungen** durchzuführen
 - ...
- In Java werden Verzweigungen durch if- und switch-Anweisungen umgesetzt.
- Verzweigungen gehören (wie später Schleifen) zu den **Kontrollstrukturen**.

Verzweigungen mit if und else

Bedingung wird zu true oder false ausgewertet

```
      ┌───┐  
if (x == 5)  
{  
    <Anweisungen1>  
}  
else  
{  
    <Anweisungen2>  
}
```

- Wenn die Variable den Wert 5 hat (also bei true), werden die **Anweisungen** im Block nach dem `if` ausgeführt.
- Falls die Variable einen Wert verschieden von 5 hat (also bei false), werden die **Anweisungen** im Block nach dem `else` ausgeführt.
- Der `else`-Zweig ist optional, d.h. eine `if`-Anweisung kann auch ohne `else` verwendet werden.

Verzweigungen mit if und else - Beispiel

- Beispiel: Ausschnitt Filmbestellung...

```
System.out.println("Bitte geben Sie Ihr Alter ein: ");
Scanner scanner = new Scanner(System.in);
int alter = scanner.nextInt();

if (alter >= 18)
{
    System.out.println("Sie werden zur Bestellung weitergeleitet.");
    ...
}
else
{
    System.out.println("Der Film ist erst ab 18 Jahren freigegeben.");
    ...
}
```

Verzweigungen mit if, else if, else

```
if (x < 1)
{
    <Anweisungen1>
}
else if (x == 1)
{
    <Anweisungen2>
}
else if (x == 2)
{
    <Anweisungen3>
}
else //äquivalent zu else if (x > 2)
{
    <Anweisungen4>
}
```

- Falls x kleiner als 1 ist, wird **Anweisungen1** ausgeführt.
- Wenn x nicht kleiner als 1 aber gleich 1 ist, wird **Anweisungen2** ausgeführt.
- Wenn die beiden ersten Fälle nicht eingetreten sind, aber der dritte, wird **Anweisungen3** ausgeführt.
- Wenn alle drei Fälle nicht eingetreten sind, wird **Anweisungen4** ausgeführt.

Vergleichsoperatoren

| Bedingung | Die Bedingung tritt ein (wird true), wenn |
|-----------------------------|---|
| <code>if (a == b)</code> | a gleich b |
| <code>if (a != b)</code> | a ungleich b |
| <code>if (a <= b)</code> | a kleiner als oder gleich b |
| <code>if (a >= b)</code> | a größer als oder gleich b |
| <code>if (a < b)</code> | a kleiner als b |
| <code>if (a > b)</code> | a größer als b |
| | ist. |

Was passiert bei folgendem Code?

```
double diskriminante = p*p/4.0 - q;  
if (diskriminante >= 0);  
{  
    System.out.println(Math.sqrt(diskriminante));  
}
```

Was passiert bei folgendem Code?

```
if (screenSize > 800)
    System.out.println("ScreenSize > 800");
    screenSize = 800;
```


Stringvergleich

- Strings lassen sich **nicht** durch die Vergleichsoperatoren `==`, `<`, `<=`, `>=`, `>` vergleichen.

Nehmen wir an, wir wollen 2 eingegebene Strings
miteinander vergleichen.

```
public static void main(String[] args)
{
    Scanner scanner = new Scanner(System.in);
    System.out.print("Bitte geben Sie den ersten String ein:");
    String ersterString = scanner.nextLine();
    System.out.print("Bitte geben Sie den zweiten String ein:");
    String zweiterString = scanner.nextLine();

    if (ersterString.equals(zweiterString))
        System.out.println("Beide Strings sind gleich.");
    else if (ersterString.compareTo(zweiterString) < 0)
        System.out.println("Der erste String kommt im Alphabet vor dem zweiten String.");
    else if (ersterString.compareTo(zweiterString) > 0)
        System.out.println("Der erste String kommt im Alphabet nach dem zweiten String.");
    scanner.close();
}
```

Stringvergleich

- Strings lassen sich **nicht** durch die Vergleichsoperatoren ==, <, <=, >=, > vergleichen.
- Durch equals lassen sich zwei Strings auf Gleichheit überprüfen.

```
if (ersterString.equals(zweiterString))
```

wird true, wenn der erste String und der zweite
String gleich sind, ansonsten false

Stringvergleich

- Strings lassen sich **nicht** durch die Vergleichsoperatoren `==`, `<`, `<=`, `>=`, `>` vergleichen.
- Durch `compareTo` lassen sich zwei Strings auf deren lexikographische Reihenfolge überprüfen, d.h. in welcher Reihenfolge würden sie in einem Wörterbuch kommen. Für die Ordnung wird der ASCII Code/Unicode zugrunde gelegt.

```
if (ersterString.compareTo(zweiterString) < 0)
```



- ist kleiner null, wenn der erste String vor dem zweiten String im Alphabet (Unicode) kommt
- ist gleich null, wenn beide Strings gleich sind
- ist größer null, wenn der erste String nach dem zweiten String im Alphabet (Unicode) kommt

logische Operatoren in Verzweigungen

logische Operatoren

Eine Bedingung lässt sich auch umkehren
durch ein **logisches NICHT: !**

```
if (!(c < d))  
{  
    <Anweisungen>  
}
```

Wenn c kleiner d **nicht** erfüllt ist, wird der Anweisungsblock ausgeführt.

logische Operatoren

Zwei Bedingungen lassen sich mit den logischen Operatoren **&&** und **||** verknüpfen.

```
if ((a == b) && (c < d))  
{  
    <Anweisungen1>  
}  
else  
{  
    <Anweisungen2>  
}
```

logisches UND: &&

Damit der erste Anweisungsblock ausgeführt wird, müssen **beide** Bedingungen erfüllt sein.

```
if ((a == b) || (c < d))  
{  
    <Anweisungen1>  
}  
else  
{  
    <Anweisungen2>  
}
```

logisches ODER: ||

Damit der erste Anweisungsblock ausgeführt wird, muss **eine** der Bedingungen erfüllt sein **oder beide**.

Kombination logischer Operatoren - Beispiel

```
if (!(a == b) && (c < d))  
{  
    <Anweisungen>  
}
```

Wenn a gleich b und c kleiner d, wird der Anweisungsblock **nicht** ausgeführt.

anders formuliert: Wenn a gleich b falsch oder c < d falsch ist (oder beide), wird der Block ausgeführt.

if-Anweisungen zählen als normale Anweisungen und dürfen überall dort auftreten, wo normale Anweisungen auftreten.

Daher können if-Anweisungen auch **geschachtelt** werden, d.h. if-Anweisungen können innerhalb von if-Anweisungen vorkommen:

```
if (a == b)
{
    <Anweisungen1>
    if (c < d)
    {
        <Anweisungen2>
    }
    else //d.h. c >= d
    {
        <Anweisungen3>
    }
    <Anweisungen4>
}
```

Auswertung von Ausdrücken

| Operator(en) | Priorität |
|---|-----------------|
| [] . (<parameters>) expr++ expr-- | 1 (höchste) |
| ++expr --expr +expr -expr ~ ! (<type>) new | 2 |
| * / % | 3 |
| + - | 4 |
| << >> >>> | 5 |
| < <= > >= instanceof | 6 |
| == != | 7 |
| & | 8 |
| ^ | 9 |
| | 10 |
| && | 11 |
| | 12 |
| ? : | 13 |
| = += -= *= /= %= <<= >>= >>>= &= ^= = | 14 (niedrigste) |

Unterscheidet sich der folgende Code im Ergebnis?

```
if (a <= b && a >= c)
    System.out.println("Hallo");
```

```
if ((a <= b) && (a >= c))
    System.out.println("Hallo");
```

Vergleich von Kommazahlen

- Da das Speichern von und Rechnen mit Kommazahlen nicht immer mathematisch exakt ist, kann i.d.R. die Gleichheit von Kommazahlen **nicht** wie folgt geprüft werden:

```
if (d == 3.3)  
{  
—<Anweisungen>  
}
```

Vergleich von Kommazahlen

```
double d = 4.4;  
d=d -1.1;
```

```
if (d == 3.3) System.out.println("wird nicht ausgegeben");
```

Vergleich von Kommazahlen

- Da das Speichern von und Rechnen mit Kommazahlen nicht immer mathematisch exakt ist, kann i.d.R. die Gleichheit von Kommazahlen **nicht** wie folgt geprüft werden:

```
if (d == 3.3)  
{  
—<Anweisungen>  
}
```

- sondern nur mit einer bestimmten Genauigkeit:

```
final double EPSILON = 1.0e-5; //0.00001  
if ((d - 3.3 >= -EPSILON) && (d - 3.3 <= EPSILON))  
{  
    <Anweisungen>  
}
```

Beispiel

Für den Einstieg in die 2. Qualifikationsebene bei der Bayerischen Polizei muss man unter anderem

- mind. 165cm groß sein
- zwischen 17 und 25 Jahre alt sein

Ausnahmen sind möglich. Je mehr Kriterien nicht erfüllt sind, desto unwahrscheinlicher eine Einstellung.

Wie können wir diese Eignungsüberprüfung in einem Programm umsetzen?

```
public class BayrischePolizeiEinstellung1  
{
```

Wir benötigen eine Klasse!

```
}
```



```
public class BayrischePolizeiEinstellung1
{
    public static void main(String[] args)
    {
```

Wir benötigen eine main-Methode!

```
    }
}
```

```
public class BayrischePolizeiEinstellung1
{
    public static void main(String[] args)
    {
```

Wir müssen Alter und Größe des
Kandidaten einlesen!

```
        Scanner scanner = new Scanner(System.in);
        System.out.print("Bitte geben Sie Ihr Alter ein: ");
        int alter = scanner.nextInt();
        System.out.print("Bitte geben Sie Ihre Größe ein: ");
        int groesse = scanner.nextInt();
```

```
        scanner.close();
    }
}
```

```
public class BayrischePolizeiEinstellung1
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Bitte geben Sie Ihr Alter ein: ");
        int alter = scanner.nextInt();
        System.out.print("Bitte geben Sie Ihre Größe ein: ");
        int groesse = scanner.nextInt();

        if ((alter > 25 || alter < 17) && groesse < 165)
            System.out.println("Eine Einstellung ist sehr unwahrscheinlich.");

        scanner.close();
    }
}
```

Wir überprüfen, welche Kriterien
von dem Kandidaten erfüllt
werden:

Werden beide Kriterien
nicht erfüllt, ist eine
Einstellung sehr
unwahrscheinlich!

```
public class BayrischePolizeiEinstellung1
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Bitte geben Sie Ihr Alter ein: ");
```

```
        int alter = scanner.nextInt();
```

```
        System.out.print("Bitte geben Sie Ihre Größe ein: ");
```

```
        int groesse = scanner.nextInt();
```

```
        if ((alter > 25 || alter < 17) && groesse < 165)
```

```
            System.out.println("Eine Einstellung ist sehr unwahrscheinlich.");
```

```
        else if (alter > 25 || alter < 17)
```

```
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
```

```
        else if (groesse < 165)
```

```
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
```

```
        scanner.close();
```

```
    }
```

```
}
```

Wird nur ein Kriterium
nicht erfüllt, ist eine
Einstellung zumindest
unwahrscheinlich.

```

public class BayrischePolizeiEinstellung1
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Bitte geben Sie Ihr Alter ein: ");
        int alter = scanner.nextInt();
        System.out.print("Bitte geben Sie Ihre Größe ein: ");
        int groesse = scanner.nextInt();

        if ((alter > 25 || alter < 17) && groesse < 165)
            System.out.println("Eine Einstellung ist sehr unwahrscheinlich.");
        else if (alter > 25 || alter < 17)
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
        else if (groesse < 165)
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
        else
            System.out.println("Eine Einstellung ist möglich.");
        scanner.close();
    }
}

```

Werden beide Kriterien
erfüllt, ist eine Einstellung
zumindest möglich.

```

public class BayrischePolizeiEinstellung1
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Bitte geben Sie Ihr Alter ein: ");
        int alter = scanner.nextInt();
        System.out.print("Bitte geben Sie Ihre Größe ein: ");
        int groesse = scanner.nextInt();

        if ((alter > 25 || alter < 17) && groesse < 165)
            System.out.println("Eine Einstellung ist sehr unwahrscheinlich.");
        else if (alter > 25 || alter < 17)
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
        else if (groesse < 165)
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
        else
            System.out.println("Eine Einstellung ist möglich.");
        scanner.close();
    }
}

```

An diesen Stellen haben wir **Code doppelt verwendet**.

```
public class BayrischePolizeiEinstellung1
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Bitte geben Sie Ihr Alter ein: ");
```

```
        int alter = scanner.nextInt();
```

```
        System.out.print("Bitte geben Sie Ihre Größe ein: ");
```

```
        int groesse = scanner.nextInt();
```

```
        if ((alter > 26 || alter < 17) && groesse < 165)
```

```
            System.out.println("Eine Einstellung ist sehr unwahrscheinlich.");
```

```
        else if (alter > 26 || alter < 17)
```

```
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
```

```
        else if (groesse < 165)
```

```
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
```

```
        else
```

```
            System.out.println("Eine Einstellung ist möglich.");
```

```
        scanner.close();
```

```
    }
```

```
}
```

Wenn wir bspw. erst ein
Alter von über 26
erlauben würden, müssten
wir an zwei Stellen
unseren Code ändern.

Oft vergisst man die Änderungen (alter von 25 auf 26)
an allen Stellen durchzuführen.



Dadurch entstehen Fehler!

Um Code gut wartbar zu machen und die Fehleranfälligkeit gering zu halten, folgt man dem **Don't Repeat Yourself (DRY)-Prinzip**



Es geht darum in einem Programm zu vermeiden, mehrfach denselben Code zu verwenden.

```
public class BayrischePolizeiEinstellung1
{
```

```
    public static void main(String[] args)
    {
```

```
        Scanner scanner = new Scanner(System.in);
        System.out.print("Bitte geben Sie Ihr Alter ein: ");
        int alter = scanner.nextInt();
        System.out.print("Bitte geben Sie Ihre Größe ein: ");
        int groesse = scanner.nextInt();
```

```
        if ((alter > 26 || alter < 17) && groesse < 165)
            System.out.println("Eine Einstellung ist sehr unwahrscheinlich.");
        else if (alter > 26 || alter < 17)
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
        else if (groesse < 165)
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
        else
            System.out.println("Eine Einstellung ist möglich.");
        scanner.close();
```

```
    }
}
```

Wie können wir die
doppelte Verwendung des
Ausdrucks vermeiden?

Speichern der Bedingung
in einer Variablen!

primitiver Datentyp: boolean

- Die Auswertung komplexerer logischer Bedingungen kann in einer Variablen vom Typ **boolean** **zwischengespeichert** werden.
- Die in einer bool'schen Variablen gespeicherten Werte werden i.d.R. in einer späteren if-Anweisung (oder Schleife) verwendet.
- Ein bool'sche Variable speichert entweder den Wert `true` oder den Wert `false`.

```
boolean bool = false;
```

```
int a = 5;
```

```
int b = 9;
```

Schlüsselwort
und
Boolean-Literal

```
System.out.println(bool);
```

```
bool = a < b;
```

```
System.out.println(bool);
```

```

public class BayrischePolizeiEinstellungDRY2
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Bitte geben Sie Ihr Alter ein: ");
        int alter = scanner.nextInt();
        System.out.print("Bitte geben Sie Ihre Größe ein: ");
        int groesse = scanner.nextInt();
        boolean alterNichtOk = alter > 26 || alter < 17;
        if (alterNichtOk && groesse < 165)
        {
            System.out.println("Eine Einstellung ist sehr unwahrscheinlich.");
        }
        else if (alterNichtOk)
        {
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
        }
        else if (groesse < 165)
        {
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
        }
        else
        {
            System.out.println("Eine Einstellung ist möglich.");
        }

        scanner.close();
    }
}

```

Wie können wir die doppelte Verwendung des Ausdrucks vermeiden?

Speichern der Bedingung in einer Variablen!

```
public class BayrischePolizeiEinstellungDRY2
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Bitte geben Sie Ihr Alter ein: ");
```

```
        int alter = scanner.nextInt();
```

```
        System.out.print("Bitte geben Sie Ihre Größe ein: ");
```

```
        int groesse = scanner.nextInt();
```

```
        boolean alterNichtOk = alter > 26 || alter < 17;
```

```
        if (alterNichtOk && groesse < 165)
```

```
            System.out.println("Eine Einstellung ist sehr unwahrscheinlich.");
```

```
        else if (alterNichtOk)
```

```
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
```

```
        else if (groesse < 165)
```

```
            System.out.println("Eine Einstellung ist unwahrscheinlich.");
```

```
        else
```

```
            System.out.println("Eine Einstellung ist möglich.");
```

```
        scanner.close();
```

```
    }
```

```
}
```

Gibt es noch mehr
doppelten Code?

Wahrheitstabellen

- Bool'sche Variablen lassen sich – wie Zahlen – auch mit == und != auswerten. Zusammenfassend als Wahrheitstabelle:

| a | b | a && b | a b | a != b | a == b |
|-------|-------|--------|--------|--------|--------|
| false | false | false | false | false | true |
| false | true | false | true | true | false |
| true | false | false | true | true | false |
| true | true | true | true | false | true |

| a | !a |
|-------|-------|
| false | true |
| true | false |

Auswertung von Ausdrücken

| Operator(en) | Priorität |
|--|-----------------|
| [] . (<i><parameters></i>) expr++ expr-- | 1 (höchste) |
| ++expr --expr +expr -expr ~ ! (<i><type></i>) new | 2 |
| * / % | 3 |
| + - | 4 |
| << >> >>> | 5 |
| < <= > >= instanceof | 6 |
| == != | 7 |
| & | 8 |
| ^ | 9 |
| | 10 |
| && | 11 |
| | 12 |
| ? : | 13 |
| = += -= *= /= %= <<= >>= >>>= &= ^= = | 14 (niedrigste) |

Welche Ausgabe ergibt folgender Code?

```
int a = 4;
int b = 15;
int c = 5;
if (a < 5 || b < 10 && c != 5)
{
    System.out.println("if-Zweig");
}
else System.out.println("else-Zweig");
```

Welche Ausgabe ergibt folgender Code?

```
int a = 4;
int b = 15;
int c = 5;
if ((a < 5 || b < 10) && c != 5)
{
    System.out.println("if-Zweig");
}
else System.out.println("else-Zweig");
```



Aufgabe

- a, b und c seien Bool'sche Ausdrücke. Erstellen Sie eine Wahrheitstabelle für folgende Ausdrücke:
 - 1) $a \parallel b \ \&\& \ c$
 - 2) $(a \parallel b) \ \&\& \ c$
- Für welche Belegungen von a, b und c unterscheiden sich die Tabellen?

Bedingte Anweisung durch Bedingungsoperator

- Eine if-else Anweisung lässt sich oft auch durch einen Bedingungsoperator (ternären Operator (**?** :)) ausdrücken.

Operand **Operand** **Operand**



<Variable> = <bool'scher Ausdruck> ? <>true-Ausdruck> : <>false-Ausdruck>

Vergleich if-else und Bedingungsoperator

if-else Anweisung

```
Scanner scanner =  
    new Scanner(System.in);
```

```
int x = scanner.nextInt();  
int y = scanner.nextInt();  
int max;
```

```
if (x > y) max = x;  
else max = y;
```

Bedingungsoperator

```
Scanner scanner =  
    new Scanner(System.in);
```

```
int x = scanner.nextInt();  
int y = scanner.nextInt();  
int max;
```

```
max = (x > y) ? x : y;
```

Auswertung von Ausdrücken

| Operator(en) | Priorität |
|---|-----------------|
| [] . (<i><parameters></i>) expr++ expr-- | 1 (höchste) |
| ++expr --expr +expr -expr ~ ! (<i><type></i>) new | 2 |
| * / % | 3 |
| + - | 4 |
| << >> >>> | 5 |
| < <= > >= instanceof | 6 |
| == != | 7 |
| & | 8 |
| ^ | 9 |
| | 10 |
| && | 11 |
| | 12 |
| ? : | 13 |
| = += -= *= /= %= <<= >>= >>>= &= ^= = | 14 (niedrigste) |

Was ergibt folgender Ausdruck?
max = x > y ? x : y;

mathematische Definition der Vorzeichenfunktion

- Mathematische Funktionen können mit Hilfe einer Programmiersprache nachgebildet werden.

Definition:

$$\text{sign}: \mathbb{R} \rightarrow \{-1, 0, 1\}$$

$$\text{sign}(x) := \begin{cases} -1, & x < 0 \\ 1, & x > 0 \\ 0, & x = 0 \end{cases}$$

Beispiele:

$$\text{sign}(-5) = -1$$

$$\text{sign}(0) = 0$$

$$\text{sign}(5) = 1$$

if-else - Beispiel

Das Vorzeichen einer eingegebenen Zahl kann wie folgt bestimmt werden:

```
Scanner scanner = new Scanner(System.in);  
int zahl = scanner.nextInt();  
  
int vorzeichen = 0;  
if (zahl < 0) vorzeichen = -1;  
else if (zahl > 0) vorzeichen = 1;  
  
System.out.println(vorzeichen);
```

mathematische Definition des Betrags

Definition:

$$|\cdot| : \mathbb{R} \rightarrow \mathbb{R}_+$$
$$x \mapsto \begin{cases} x, & x \geq 0 \\ -x, & x < 0 \end{cases}$$

Beispiele:

$$|5| = 5$$

$$|-5| = 5$$

$$|0| = 0$$

Übungen zum Betrag

Sei $x, y \in \mathbb{R}$ beliebig.

Zeigen Sie (mit Hilfe der Betragsdefinition):

1. $|xy| = |x| \cdot |y|$

2. $|x + y| \leq |x| + |y|$

Übungen zum Betrag

Sei $x, y \in \mathbb{R}$ beliebig.

Zeigen Sie (mit Hilfe der Betragsdefinition):

1. $|xy| = |x| \cdot |y|$

2. $|x + y| \leq |x| + |y|$