

## Fenster mit Menü ausstatten

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BeispielMenu extends BasisFenster {
    public BeispielMenu() {
        super("Beispiel Menu", 500, 100);
        setMenuBar(getMenubar());
        setVisible(true);
    }

    private void start() {
        System.out.println("Start");
    }
}
```

## Fenster mit Menü ausstatten II

```
private MenuBar getMenubar() {  
    MenuBar menueLeiste = new MenuBar();  
    Menu spiel = new Menu("SPIEL"); // Erste Menüspalte SPIEL  
  
    // Erster Menüpunkt Start  
    MenuItem start = new MenuItem("Start");  
    start.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {  
            start();  
        }  
    });  
    spiel.add(start);  
    // Zweiter Menüpunkt Ende  
    spiel.add(new MenuItem("Ende"));  
    // Trennlinie  
    spiel.addSeparator();  
    // Dritter Menüpunkt Credits  
    spiel.add(new MenuItem("Credits"));  
    menueLeiste.add(spiel);  
    // Zweite Menüspalte OPTIONEN  
    Menu optionen = new Menu("OPTIONEN");  
    // Erster Menüpunkt Musik mit Verzweigung  
    Menu musik = new Menu("Musik");  
    optionen.add(musik);  
    musik.add("An");  
    musik.add("Aus");  
    menueLeiste.add(optionen);  
  
    return menueLeiste;  
}
```

## Fenster mit Menü ausstatten III



## Live-Coding-Session

```
number": $number,
contents": $contents,
$count": $count,

$i + 1' . $totalSecurity)
echo "checked";
if ($i == 0) {
    ("checked");
}
```

## Mehrere Optionen oder exklusive Auswahl



## Live-Coding-Session

```
number": $number,
contents": $contents,
$count": $count,

$i + 1' . $totalSecurity)
echo ($i == 0) ?
("checked"),
```

## Speichern der Benutzereinstellungen

Die vorgestellte Methode, Daten in Dateien zu speichern, haben wir erfolgreich beim Fußballmanagerprojekt eingesetzt. Das wäre auch für das Speichern von Benutzereinstellungen in selbst angelegten Konfigurationsdateien eine gute Alternative.

Wir werden trotzdem einen kleinen Blick auf die Klasse Preferences werfen, die in einem solchen Kontext sehr oft verwendet wird und dem Programmierer viel Arbeit abnehmen kann.

## Konfigurationen speichern mit Preferences

Angenommen, wir haben in unserer Klasse PreferencesTest eine Zeichenkette name vorliegen und wollen diese in einer Konfigurationsdatei speichern und bei Bedarf wieder laden:

```
public class PreferencesTest {  
    private String name;  
  
    // get- und set-Methoden  
    ...  
}
```

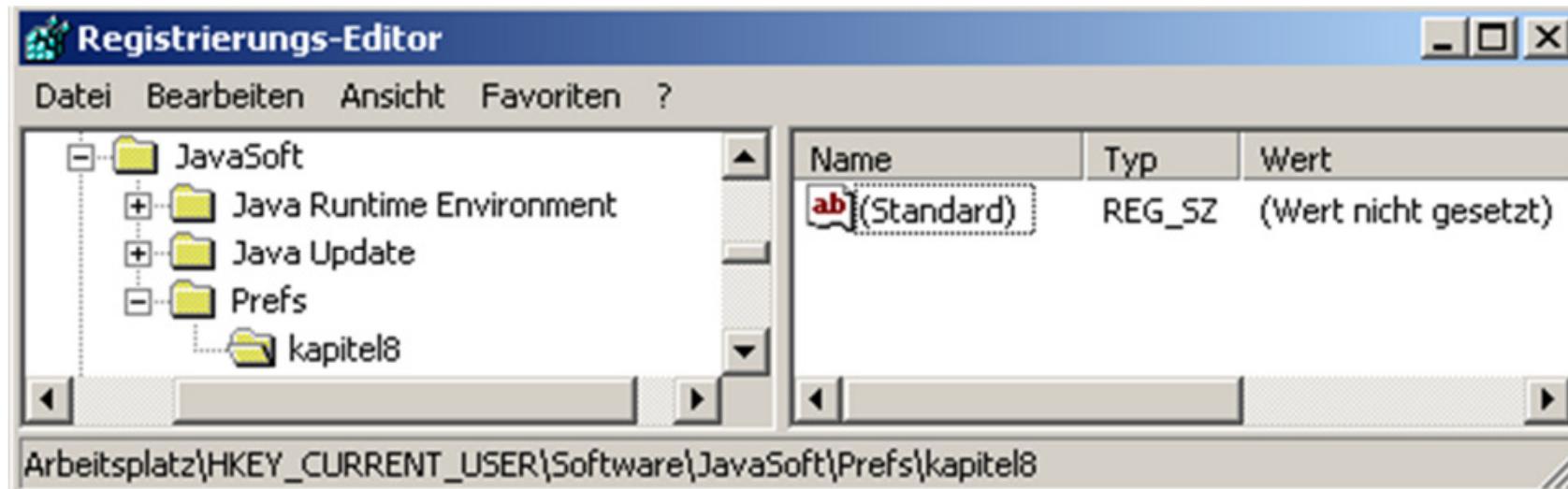
Dazu legen wir uns ein Exemplar der Klasse Preferences an:

```
private Preferences prefs = Preferences.userNodeForPackage(getClass());
```

Über die Methode getClass können wir den Namen der vorliegenden Klasse in Erfahrung bringen. So erhalten wir beispielsweise den Namen und Ort der aktuellen Klasse class kapitel8.PreferenceTest.

In Windows wird für den identifizierten Packagennamen über die Methode userNodeForPackage in der Registry ein neuer Zweig in SOFTWARE/JavaSoft/Prefs unter dem Eintrag HKEY\_CURRENT\_USER angelegt.

## Konfigurationen speichern mit Preferences II



## Konfigurationen speichern mit Preferences III

Wir spendieren unserer Klasse jetzt eine Methode zum Speichern und eine zum Laden des Parameters name:

```
// Konfiguration speichern
public void setPreferences() {
    setName("Heinz");
    prefs.put("name", name);
}

// Konfiguration laden
public void getPreferences() {
    name = prefs.get("name", "");
}
```

Mit prefs.put wird eine Zeichenkette name mit dem entsprechenden Wert angelegt:



## Konfigurationen speichern mit Preferences IV

Beim Laden über prefs.get wird noch ein Defaultparameter angegeben, der gesetzt wird, falls name nicht gefunden wird.

Mit dem folgenden Beispiel können wir den Effekt gut verfolgen:

```
PreferencesTest test = new PreferencesTest();
test.getPreferences();
System.out.println("Erstes Laden: >" + test.getName() + "<");
test.setPreferences();
test.getPreferences();
System.out.println("Zweites Laden: >" + test.getName() + "<");
```

Wir erhalten nach dem erstmaligen Aufruf von getPreferences den Defaultwert für name und beim zweiten dann den über setPreferences gesetzten Wert Heinz:

```
Erstes Laden: ><
Zweites Laden: >Heinz<
```

Sollten wir das Programm ein zweites Mal ausführen, erhalten wir wie erwartet bei beiden Ausgaben den Wert Heinz:

```
Erstes Laden: >Heinz<
Zweites Laden: >Heinz<
```

Neben der Methode put, die eine Zeichenkette speichert, gibt es weitere nützliche Methoden, wie putBoolean, putInt, putLong, putFloat und putDouble mit den dazugehörigen Auslesemethoden.

## Klartextdarstellung verschleiern

Wer die Klartextdarstellung in der Konfigurationsdatei etwas verschleiern möchte, kann beispielsweise auf die Verschlüsselungsmethode zurückgreifen:

```
public void setPreferences() {  
    setName("Heinz");  
    prefs.put("name", Codierer.codiere(name, key));  
}  
  
public void getPreferences() {  
    name = Codierer.codiere(prefs.get("letzte Eingabe", ""), key);  
}
```

Wir wissen allerdings, dass die gewählte Verschlüsselungsmethode bei größeren Textmengen leicht zu entschlüsseln ist. Hier allerdings ist sie ein probates Mittel.

## Fenstergröße erhalten

Als abschließendes Beispiel wollen wir ein Fenster erzeugen, das seine Größe über eine Konfigurationsdatei speichert und diese bei einem erneuten Start laden kann:

```
import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.prefs.Preferences;

public class FensterKonfigurieren extends Frame {
    private Preferences prefs = Preferences.userNodeForPackage(getClass());
    private int breite, hoehe;

    public FensterKonfigurieren(String titel) {
        setTitle(titel);
        // Konfiguration auslesen
        breite = prefs.getInt("breite", 500);
        hoehe = prefs.getInt("hoehe", 400);
        setSize(breite, hoehe);
        setLocationRelativeTo(null);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                // Konfiguration speichern
                prefs.putInt("breite", getWidth());
                prefs.putInt("hoehe", getHeight());

                event.getWindow().dispose();
                System.exit(0);
            }
        });
        setVisible(true);
    }
}
```

## Live-Coding-Session

```
number": $number,
contents": $contents,
$count": $count,
 $i + 1,
echo "checked";
if ($i == 0) {
    ("checked");
}
```

## Die Klasse Color

Um das RGB-Farbmodell zu verwenden, erzeugen wir viele farbige Rechtecke, deren 3 Farbkomponenten rot, grün und blau zufällig gesetzt werden. Dazu erzeugen wir ein Color-Objekt und setzen dessen Farbwerte:

```
import java.awt.*;
import java.util.Random;

public class FarbBeispiel extends Frame {
    public FarbBeispiel(String titel) {
        setTitle(titel);
        setSize(500, 300);
        setBackground(Color.lightGray);
        setForeground(Color.red);
        setVisible(true);
    }

    public void paint(Graphics g){
        Random r = new Random();
        for (int y=30; y<getHeight()-10; y += 15)
            for (int x=12; x<getWidth()-10; x += 15) {
                g.setColor(new Color(r.nextInt(256), r.nextInt(256), r.nextInt(256)));
                g.fillRect(x, y, 10, 10);
                g.setColor(Color.BLACK);
                g.drawRect(x - 1, y - 1, 10, 10);
            }
    }

    public static void main(String[] args) {
        FarbBeispiel t = new FarbBeispiel("Viel Farbe im Fenster");
    }
}
```

## Ausgabe



# Live-Coding-Session

```
    "number": $number,
    "contents": $contents,
    "put $count: $1++),
    $i + 1, type="/" ),
    $totalSecurity);
    $i == 0) {
    ("checked"),
    ("checked"),
```

## Bilder laden und anzeigen

Mit der Methode `drawImage` können wir Bilder anzeigen lassen. In den Zeilen 14 bis 16 geschieht aber leider nicht das, was wir erwarten würden. Die Funktion `getImage` bereitet das Laden des Bildes nur vor. Der eigentliche Ladevorgang erfolgt erst beim Aufruf von `drawImage`.

Das hat den Nachteil, dass bei einer Wiederverwendung der Methode `paint` jedes Mal das Bild neu geladen wird:

```
import java.awt.*;  
  
public class BildFenster extends Frame {  
    public BildFenster(String titel) {  
        setTitle(titel);  
        setSize(423, 317);  
        setBackground(Color.lightGray);  
        setForeground(Color.red);  
        setVisible(true);  
    }  
  
    public void paint(Graphics g){  
        Image pic = Toolkit.getDefaultToolkit().getImage("C:\\\\kreidefelsen.jpg");  
        g.drawImage(pic, 0, 0, this);  
    }  
  
    public static void main( String[] args ) {  
        BildFenster t = new BildFenster("Bild im Fenster");  
    }  
}
```

Jetzt könnten wir sogar schon eine Diashow der letzten Urlaubsbilder realisieren.

## Ausgabe



## Live-Coding-Session

```
number": $number,
contents": $contents,
$count": $count,

$i + 1' . $totalSecurity)
echo ($i == 0) ?
("checked"),
```

## Der MediaTracker

Eine globale Variable img vom Typ Image wird angelegt und im Konstruktor mit dem Mediatracker verknüpft. Der Mediatracker liest die entsprechenden Bilder ein und speichert sie:

```
// wird z.B. dem Konstruktor hinzugefügt
img = getToolkit().getImage("c:\\kreidefelsen.jpg");

MediaTracker mt = new MediaTracker(this);
mt.addImage(img, 0);
try {
    mt.waitForAll();
} catch (InterruptedException e){
    e.printStackTrace();
}
```

Jetzt können wir in der paint-Methode mit dem folgenden Aufruf das Bild aus dem Mediatracker laden und anzeigen:

```
g.drawImage(img, 0, 0, this);
```

Falls wir das Bild noch skalieren wollen, fügen wir vor dem Einlesen über den MediaTracker noch diese Zeile ein:

```
imgScaled = img.getScaledInstance(WIDTH, HEIGHT, Image.SCALE_SMOOTH);
```

## Auf Mausereignisse reagieren

Im folgenden Beispiel wollen wir für einen Mausklick innerhalb des Fensters einen grünen Punkt an die entsprechende Stelle zeichnen. Hier wird einfach das Interface MouseListener implementiert oder die leeren Methoden der Klasse MouseAdapter überschrieben:

```
import java.awt.*;
import java.awt.event.*;

public class MausKlick extends MeinFenster {
    public MausKlick(String titel, int w, int h) {
        super(titel, w, h);
        setSize(w, h);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                Graphics g = getGraphics();
                g.setColor(Color.green);
                g.fillOval(e.getX(), e.getY(), 10, 10);
            }
        });
        setVisible(true);
    }

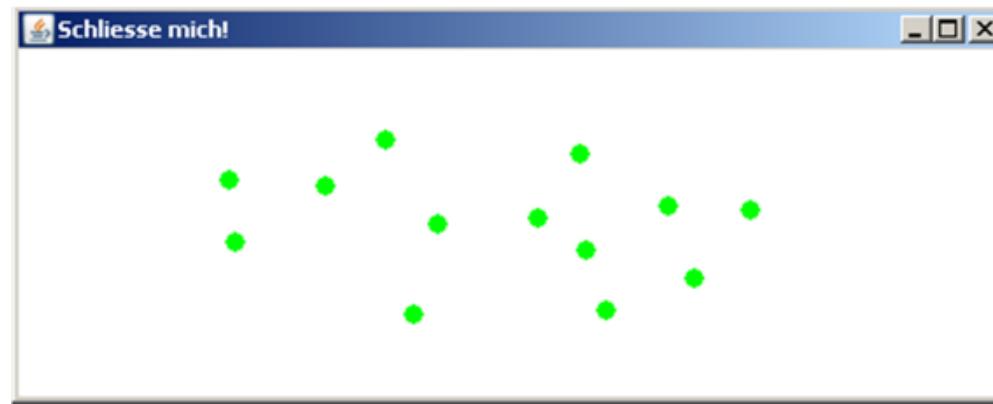
    public static void main(String[] args) {
        MausKlick f = new MausKlick("Schliesse mich!", 500, 200);
    }
}
```

## Live-Coding-Session

```
number": $number,
contents": $contents,
$count": $count,

$i + 1' . $totalSecurity)
echo ($i == 0) ?
("checked"),
```

## Ausgabe



## Double buffering

```
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import javax.swing.JComponent;

public class Anzeige extends JComponent {
    private static final long serialVersionUID = 1L;
    private BufferedImage front;

    public Anzeige(BufferedImage front) {
        this.front = front;
        this.setPreferredSize(new Dimension(front.getWidth(), front.getHeight()));
    }

    public BufferedImage flip(BufferedImage back) {
        BufferedImage newBack = front;
        synchronized (this) {
            front = back;
        }
        repaint();
        return newBack;
    }

    public BufferedImage getFrontImage() {
        return front;
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        synchronized (this) {
            g.drawImage(front, 0, 0, this);
        }
    }
}
```

## FullScreen in Java

```
...
public AnalogUhrVollbild() {
    // Vollbildanzeige
    ANZEIGE_BREITE = (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
    ANZEIGE_HOEHE = (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
    setSize(ANZEIGE_BREITE, ANZEIGE_HOEHE);
    setUndecorated(true);

    // mit Escape lässt sich das Fenster schließen
    addKeyListener(new KeyAdapter() {
        public void keyPressed(KeyEvent evt) {
            if (evt.getKeyCode() == KeyEvent.VK_ESCAPE)
                System.exit(0);
        }
    });
}

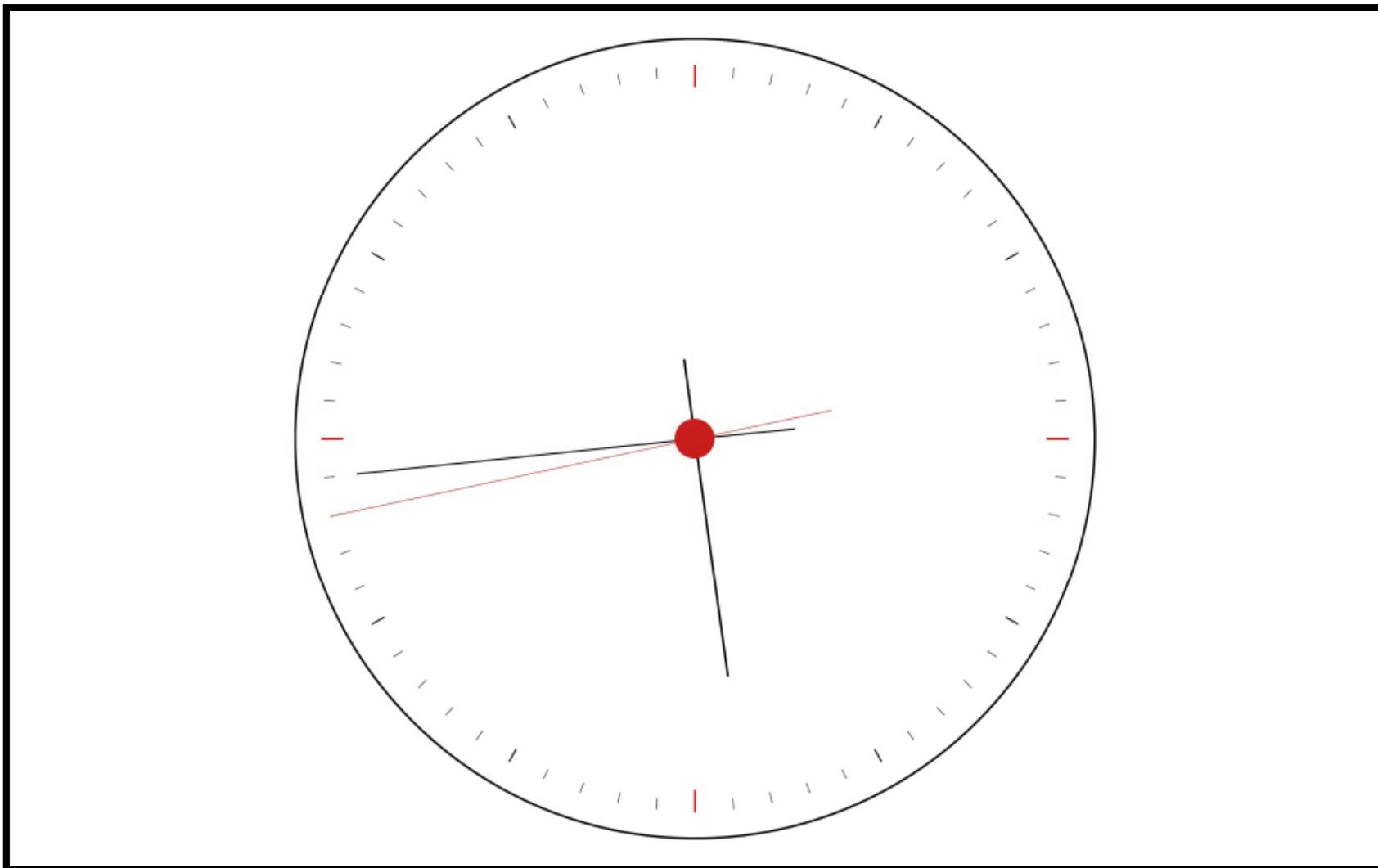
// Renderscreen wird angelegt
offscreen = new BufferedImage(ANZEIGE_BREITE, ANZEIGE_HOEHE,
    BufferedImage.TYPE_INT_RGB);
onscreen = new BufferedImage(ANZEIGE_BREITE, ANZEIGE_HOEHE,
    BufferedImage.TYPE_INT_RGB);
anzeige = new Anzeige(onscreen);
getContentPane().add(anzeige);

// Uhr wird mit zentral und möglichst groß angelegt
uhr = new AnalogUhr(new Point2D.Double(ANZEIGE_BREITE/2, ANZEIGE_HOEHE/2),
    ANZEIGE_HOEHE/2 - 40);

pack();
setVisible(true);
}
...
```

## Projekt: Analoge Uhr

Swing, Double buffering, fullscreen, Threads



## Live-Coding-Session

```
number": $number,
contents": $contents,
$count": $count,

$i + 1' . $totalSecurity)
echo ($i == 0) ?
("checked"),
```

## Simulation einer bunten Konsole

```
public class AnzeigeKonsole extends JFrame {  
    ...  
    public int getDatenBreite() ...  
    public int getDatenHoehe() ...  
    public void setDaten(Zeichen[][] daten) ...  
    public void setHintergrundFarbe(Color hintergrundFarbe) ...  
    ...  
}
```

## Symbolklasse

```
import java.awt.Color;

public class Zeichen {
    private Color farbe;
    private char symbol;

    public Zeichen() {
        setFarbe(Color.WHITE);
        setSymbol('_');
    }

    public Zeichen(Color farbe, char symbol) {
        setFarbe(farbe);
        setSymbol(symbol);
    }

    public Color getFarbe() {
        return farbe;
    }

    public void setFarbe(Color farbe) {
        this.farbe = farbe;
    }

    public char getSymbol() {
        return symbol;
    }

    public void setSymbol(char symbol) {
        this.symbol = symbol;
    }
}
```

# Projekt: Conway's Game of Life



## Live-Coding-Session

```
number": $number,
contents": $contents,
$count": $count,

$i + 1' . $totalSecurity)
echo ($i == 0) ?
("checked"),
```

## Bilder transformieren

```
...
// "Symbol"-Farbtiefe
private char[] pixelWert = { ' ', '.', ',', ':',
                             ';', 'i', 'c', 'o',
                             'ö', '%', '&', '$',
                             '§', 'X', '#', 'M'};

...
public void anzeigen() {
    W = anzeige.getDatenBreite();
    H = anzeige.getDatenHoehe();
    System.out.println("W:" + W + " H:" + H);

    zeichenBild = new Zeichen[W] [H];
    for (int x=0; x<W; x++) {
        for (int y=0; y<H; y++) {
            int farbe      = bild.getRGB(x, y);
            int rot       = (farbe & 256*256*255)/(256*256);
            int gruen     = (farbe & 256*255)/256;
            int blau       = farbe & 255;
            int grauwert   = (int) Math.floor(0.299*rot + 0.587*gruen + 0.114*blau);

            int bitAnteil = 256/pixelWert.length;
            zeichenBild[x][y] = new Zeichen(new Color(farbe),
                                         pixelWert[(255-grauwert)/bitAnteil]);
        }
    }

    anzeige.setDaten(zeichenBild);
    anzeige.repaint();
}
...
```

# Bilder einlesen und in Symbolik transformieren

## Live-Coding-Session

```
number": $number,
contents": $contents,
$count": $count,

$i + 1' . $totalSecurity)
echo ($i == 0) ?
("checked"),
```

Vorlesungsteil

## Einführung in die Algorithmik



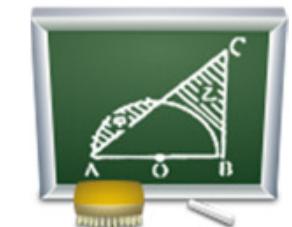
*Es gibt keinen Königsweg zur Mathematik.  
Euklid*

# Übersicht zum Vorlesungsinhalt

zeitliche Abfolge und Inhalte können variieren

## Techniken der Programmentwicklung

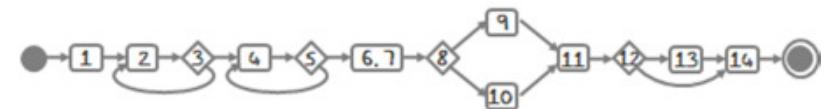
- Landau-Symbolik
- ...



## Definition Algorithmus

Mit Algorithmen beschreiben wir genau definierte Handlungsvorschriften zur Lösung eines Problems oder einer bestimmten Art von Problemen. Es geht darum, eine genau spezifizierte Abfolge von Anweisungen auszuführen, um ein gegebenes Problem zu lösen.

Zutaten:	Zubereitung:
4 Eier	- vier Eier in eine Schüssel schlagen und verrühren
Mehl	- solange Mehl hinzugeben, bis der Teig schwer zu rühren ist
Milch	- solange Milch hinzugeben, bis die Masse wieder leicht zu rühren ist,
1 Apfel	damit sie sich gut in der Pfanne verteilen lässt
Wurst/Käse	- etwas Fett in einer Pfanne erhitzen
Marmelade	- Teil der Masse in die Mitte der Pfanne geben, bis der Boden bedeckt ist
Apfelmus	- bei der süßen Variante können Apfelscheiben dazu gegeben werden, bei der herzhaften Variante Wurst und Käse
Zucker	- die Eierpfannkuchen von beiden Seiten gut anbraten - die süßen Eierpfannkuchen können nach belieben, z. B. mit Marmelade, Apfelmus oder Zucker garniert werden



## Algorithmus versus Programm

Für Algorithmen gibt es unterschiedliche Möglichkeiten der formalen Repräsentation. Von einer Universalsprache (Pseudocode) bis hin zu konkreten Maschinenbefehlen reicht der Begriff Programm.

Auch Eingaben und Konfigurationen für abstrakte Computermodelle, wie bei der Turingmaschine werden als Programme bezeichnet.

```
function timestamp_mysql($dateex) {
    $dateex = explode('-', $dateex);
    $year = substr($dateex[2], 2, 2);
    $month = substr($dateex[1], 2, 2);
    $day = substr($dateex[0], 2, 2);
    return sprintf("%04d-%02d-%02d %02d:%02d:%02d");
}

function date_mysql2ger($dateex) {
    list($year, $month, $day) =
        sscanf($dateex, "%04d-%02d-%02d %02d:%02d:%02d");
    $year = substr($year, 2, 2);
    $month = substr($month, 2, 2);
    $day = substr($day, 2, 2);
    return sprintf("%02d.%02d.%02d-%02d.%02d.%02d %02d.%02d.%02d");
}

function date_ger2mysql($dateex) {
    list($day, $month, $year) =
        sscanf($dateex, "%02d.%02d.%02d-%02d.%02d.%02d %02d.%02d.%02d");
    $year = substr($year, 2, 2);
    $month = substr($month, 2, 2);
    $day = substr($day, 2, 2);
    return sprintf("%04d-%02d-%02d %02d:%02d:%02d");
}
```

Wie definiert sich eine Primzahl?

## Primzahlen

### **Definition Primzahl**

Eine Primzahl  $p \in \mathbb{N}$  hat genau zwei ganzzahlige Teiler 1 und  $p$ .

Aus der Definition ergibt sich die Folge der Primzahlen:

2, 3, 5, 7, 11, 13, 17, 19, 23, ...

Die aktuell größte Primzahl  $2^{57.885.161} - 1$  mit 17.425.170 Stellen wurde von einem seit 17 Jahren laufenden Supercomputer (GIMPS-Projekt) im Februar 2013 ausgespuckt.

## Sieb des Eratosthenes

Einen Algorithmus zur Ermittlung der Primzahlen von 2 bis  $n$  formulierte Eratosthenes:

**sieb (n)**

notiere aufsteigend alle natürlichen Zahlen von 1 bis  $n$

**k:=1**

**do**

$m:=$ erste nicht markierte Zahl größer als  $k$

markiere alle Vielfachen von  $m$ , ausser  $m$  selbst

**k:=m**

**while** ( $k < \text{sqrt}(n)$ )

Schauen wir uns mal ein Beispiel an ...

## Beispiel n=40

k=1

m=2

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40

## Beispiel n=40

k=1

m=2

<del>1</del>	<b>2</b>	3	<del>4</del>	5	<del>6</del>	7	<del>8</del>	9	<del>10</del>
11	<del>12</del>	13	<del>14</del>	15	<del>16</del>	17	<del>18</del>	19	<del>20</del>
21	<del>22</del>	23	<del>24</del>	25	<del>26</del>	27	<del>28</del>	29	<del>30</del>
31	<del>32</del>	33	<del>34</del>	35	<del>36</del>	37	<del>38</del>	39	<del>40</del>

## Beispiel n=40

k=2

m=3

1	2	3	4	5	6	7	8	9	10
11	<del>12</del>	13	<del>14</del>	15	<del>16</del>	17	<del>18</del>	19	<del>20</del>
21	<del>22</del>	23	<del>24</del>	25	<del>26</del>	27	<del>28</del>	29	<del>30</del>
31	<del>32</del>	33	<del>34</del>	35	<del>36</del>	37	<del>38</del>	39	<del>40</del>

## Beispiel n=40

k=2

m=3

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40

Wann endet der Algorithmus?

## Beispiel n=40

k=5

m=7

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40

Was liefert der Algorithmus nach Abarbeitung?

## Beispiel n=40

k=5

m=7

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40

## Sieb des Eratosthenes

Einen Algorithmus zur Ermittlung der Primzahlen von 2 bis  $n$  formulierte Eratosthenes:

```
sieb (n)
    notiere aufsteigend alle natürlichen Zahlen von 1 bis n
    k:=1
    do
        m:=erste nicht markierte Zahl größer als k
        markiere alle Vielfachen von m, ausser m selbst
        k:=m
    while (k<sqrt(n))
```

Nach Abarbeitung des Algorithmus entsprechen alle unmarkierten Zahlen gerade den Primzahlen kleiner gleich  $n$ .

## Ein Element wird gesucht

Gegeben sei eine aufsteigend sortierte,  $n$ -elementige Folge von Elementen  $a_0, a_1, \dots, a_{n-1}$  und ein gesuchtes Element  $x$ .

Wir interessieren uns für den Index von  $x$  innerhalb der Folge, falls  $x$  darin vorkommt.

Woran erinnert uns das Suchen?

## Das Telefonbuch

Eure Eltern werden sich noch daran erinnern, als Telefone noch Kabel hatten, die Telefone nur bei der Post bestellbar waren und es neben Telefonzellen noch Telefonbücher gab:



Ich gehe mal systematisch vor ...

## Suchalgorithmus Marco

Mein erster Vorschlag:

```
suche_marco(x, a[0], a[1], ..., a[n-1])
    pos = -1
    for (i:=0 to n-1) do
        wenn a[i] gleich x
            pos = i
    endfor
    return pos
```

Mein zweiter Vorschlag:

```
suche_marco(x, a[0], a[1], ..., a[n-1])
    pos = -1
    for (i:=0 to n-1) do
        wenn a[i] gleich x
            pos = i
            beende for-Schleife
    endfor
    return pos
```

Micha?

Was würdest Du machen?

## Suchalgorithmus Micha

Micha schlägt folgendes vor:

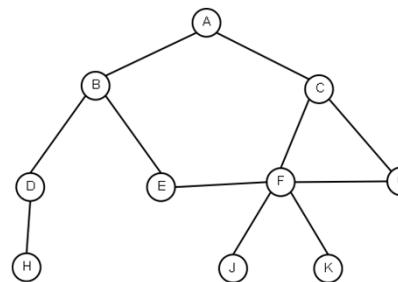
```
suche_micha(x, a[0], a[1], ..., a[n-1])
    i:=0
    j:=n-1;
    while (i<=j) do
        m:=floor((i+j)/2)
        wenn (x>a[m])
            i:=m+1
        ansonsten wenn (x<a[m])
            j:=m-1
        ansonsten
            return m
    endwhile
    return -1
```

Dieser Algorithmus ist auch als **Binäre Suche** bekannt.

## Von der Problembeschreibung bis zur Lösung

Um Probleme in der Informatik, also mit Hilfe von Computern, lösen zu können, müssen wir sie zunächst geeignet darstellen:

Symbolische **Kodierung der Informationen** bestimmen, damit diese effizient manipuliert werden können.



Nach der geeigneten Kodierung folgt die Verarbeitung durch einen Algorithmus bis zur Lösung:

Die sukzessive **Modifizierung** des Zustands durch das Abarbeiten des Algorithmus mit dem Ziel, das gewünschte Resultat zu liefern.

```
suche_micha(x, a[0], a[1], ..., a[n-1])
    i:=0
    j:=n-1;
    while (i<=j) do
        m:=floor((i+j)/2)
        wenn (x>a[m])
            i:=m+1
        ansonsten wenn (x<a[m])
            j:=m-1
        ansonsten
            return m
    endwhile
    return -1
```

## Algorithmen miteinander vergleichen

Um die Laufzeit eines Algorithmus zu analysieren, müssen wir entscheiden, auf welchem Rechnermodell dieser arbeiten soll.

Computer	Pentium IV, 1.6 Ghz, 1 GB RAM, ...
Programmiersprache	Haskell, Java, Pascal, C++, ...
Compiler	



Dann ermitteln wir die Laufzeit für bestimmte Eingaben unterschiedlicher Algorithmen und können dann in Abhängigkeit dieser entscheiden, welcher Algorithmus der schnellste ist.

Manchmal interessiert uns aber auch der günstigste im Speicherverbrauch.

Was ist nachteilig an dieser Methode?

## Registermaschine RAM

Wir benötigen ein theoretisches, abstraktes Rechnermodell. Wir wählen eine Registermaschine: die RAM (Random Access Machine) [Modell von 1963].

1. Jede einfache (atomare, elementare) Operation wie z.B.  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $=$ , if, ... benötigt nur eine Zeiteinheit bzw. einen Schritt.
2. Schleifen und Methoden sind keine elementaren Operationen, setzen sich aber aus diesen zusammen.
3. Jeder Speicherzugriff kostet genau eine Zeiteinheit.
4. Es gibt unbegrenzten Speicher.

## Komplexität von Algorithmen

Die Komplexität eines Algorithmus beschreibt dessen Kosten (Laufzeit, Speicher, Anzahl der Multiplikationen, ...). Meistens reicht der Speicher in der Praxis aus, so dass die Laufzeit das interessante Komplexitätsmaß ist.

Dieses kann anhand der Anzahl der elementaren Operationen berechnet werden. Diese hängt von der **Dimension n** der Eingabe ab:  $[x_1, x_2, \dots, x_n]$ .

Es gibt verschiedene Beschreibungsmöglichkeiten, um die Anzahl der Elementarschritte von der Größe/Dimension der Eingabe anzugeben.

## Landau-Symbolik – O-Notation

Mit der O-Notation geben wir eine obere Schranke für die Komplexität eines Algorithmus an. Wir drücken damit aus, dass ein Algorithmus nicht schlechter ist, als ein bestimmtes Maß.

Mathematisch:  $f, g :: \mathbb{N} \rightarrow \mathbb{R}^+$

Eine Funktion  $f(n)$  hat als obere Schranke  $g(n)$ , wir schreiben dann  $f(n) \in O(g(n))$ , wenn es eine positive Konstante  $n_0 \in \mathbb{N}$  gibt, so dass alle Werte von  $f(n)$  für alle  $n \geq n_0$  immer kleiner/gleich  $c \cdot g(n)$  sind.

Wir schreiben dafür kurz:

$$f(n) \in O(g(n)): \exists n_0 \in \mathbb{N}, \exists c \in \mathbb{R}^+, \forall n \geq n_0: f(n) \leq c \cdot g(n)$$

## Ein kleines Beispiel

Angenommen wir haben für einen Algorithmus  $A$  die folgende Laufzeit ermittelt:

$$A(n): 3n^4 + 5n^3 + 7\log(n)$$

Jetzt wollen wir diesen Algorithmus mit  $B(n)$  vergleichen, der aber folgende Laufzeit aufweist:

$$B(n): n^4$$

Ist A schneller als B?

Ist B schneller als A?

Sind A und B gleich schnell?

## Ein kleines „verblüffendes“ Beispiel

Angenommen wir haben für einen Algorithmus  $A$  die folgende Laufzeit ermittelt:

$$A(n): 3n^4 + 5n^3 + 7\log(n)$$

Jetzt wollen wir diesen Algorithmus mit  $B(n)$  vergleichen, der aber folgende Laufzeit aufweist:

$$B(n): n^4$$

Wir behaupten also, dass  $A(n) \in O(B(n))$

$$\text{Beweis: } 3n^4 + 5n^3 + 7\log(n) \leq 3n^4 + 5n^4 + 7n^4 \leq 15n^4$$

Ist erfüllt für  $n \geq 1$ . Daher können wir für  $c = 15$  und  $n_0 = 1$  wählen.

■

## Eine einfache Summe

Wir wollen die Summe  $\sum_{i=1}^n i$  berechnen. Dafür schreiben wir uns ein kleines Programm:

```
summe:=0
for (i:=1 to n)
    summe:=summe+i
```

Jetzt berechnen wir die Laufzeit des Programmabschnitts:

```
1 Zuw.
n Zuw. + (n-1) Zuw. + (n-1) if
    (n-1) Zuw. + (n-1) Anw.
```

Es ergibt sich also:

$$A(n): 1 + n + (n - 1) + (n - 1) + (n - 1) + (n - 1) \leq 5n$$

Eine einfachere Laufzeit, deren Äquivalenz zu  $A$  wir leicht zeigen können, hat  $B$ :

$$B(n): n$$

$$A(n) \in O(B(n)) \quad \checkmark$$

$$B(n) \in O(A(n)) \quad \checkmark$$

Beide sind gleich. Wir sprechen von einer **linearen** Laufzeit.

## Eine einfache Summe – ein anderer Blick

Wir wollen die Summe  $\sum_{i=1}^n i$  berechnen. Dafür schreiben wir uns ein kleines Programm:

```
summe:=0
for (i:=1 to n)
    summe:=summe+i
```

Jetzt berechnen wir die Laufzeit des Programmabschnitts:

```
n mal
    1
```

## Eine einfache, schnelle Summe

Wir wollen wieder die Summe  $\sum_{i=1}^n i$  berechnen. Nutzen aber die Idee von Gauss, dass  $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$  und schreiben wieder ein kleines Programm:

```
summe := n * (n+1) / 2
```

Es ergibt sich in diesem Beispiel:

$A(n)$ : 4

Eine einfachere Laufzeit, deren Äquivalenz zu  $A$  wir leicht zeigen können, hat  $B$ :

$B(n)$ : 1

$A(n) \in O(B(n))$  ✓

$B(n) \in O(A(n))$  ✓

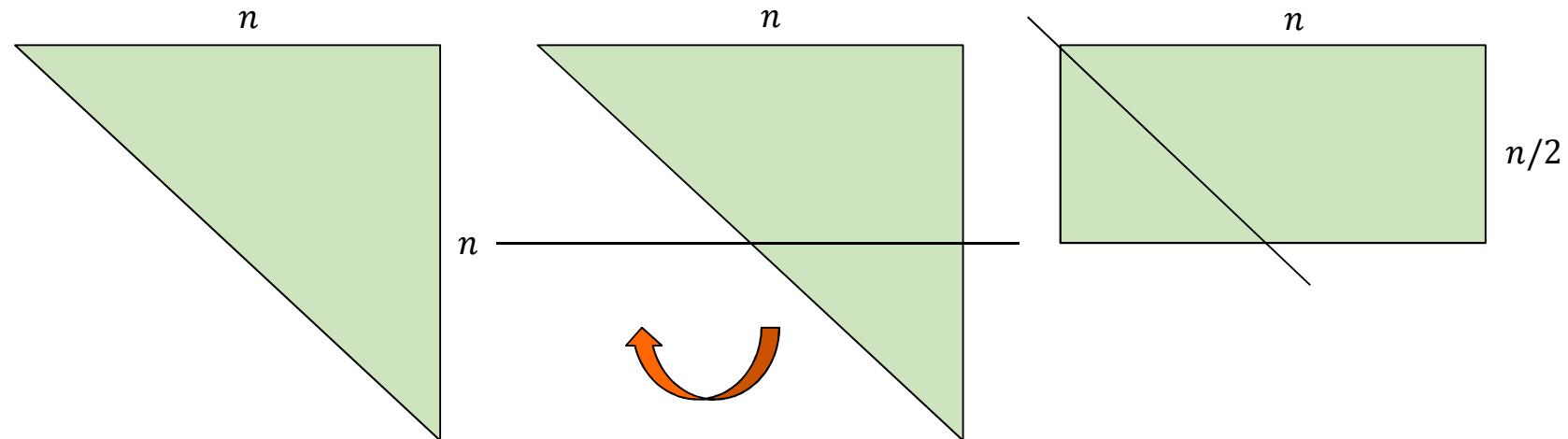
Beide sind gleich. Wir sprechen von einer **konstanten** Laufzeit.

## Eine verschachtelte Summe

Schauen wir uns folgendes Programm an:

```
summe:=0
for (i:=1 to n)
    for (j:=1 to i)
        summe:=summe+1
```

Visuelle Interpretation:



Es ergibt sich in diesem Beispiel:

$$A(n): n \cdot \frac{n}{2}$$

Eine einfachere Laufzeit, deren Äquivalenz zu  $A$  wir leicht zeigen können, hat  $B$ :

$$B(n): n^2$$

$$A(n) \in O(B(n)) \quad \checkmark$$

$$B(n) \in O(A(n)) \quad \checkmark$$

Beide sind gleich. Wir sprechen von einer **quadratischen** Laufzeit.

## Optimalität

Ein Algorithmus für ein gegebenes Problem ist **optimal**, wenn seine Komplexität eine untere Schranke für die Komplexitäten anderer Algorithmen darstellt, die das Problem ebenfalls lösen.

$A(n)$  ist optimal  $\Leftrightarrow A(n) \in O(B(n)) \wedge A(n) \in O(C(n)) \wedge A(n) \in O(D(n))$ , mit  
 $B(n)$ ,  $C(n)$  und  $D(n)$  sind weitere Lösungen



## Wichtige Komplexitätsklassen

Zu den wichtigen Komplexitätsklassen gehören:

Funktion	Beispiel	Typische Bezeichnung
$1$	Nachsehen in einem Array	konstant
$\log n$	Teile-und-Herrsche	logarithmisch
$\log \log n$	Suche in sortierten Arrays	-
$\log^k n$	-	polylogarithmisch
$\sqrt{n}$	Naiver Primzahltest	-
$n$	Maximum einer Liste bestimmen	linear
$n \log n$	Vergleichsbasiertes Sortieren	-
$n^2$	Alle Paare einer Grundmenge ansehen	quadratisch
$n^3$	Lineare Gleichungssysteme lösen	kubisch
$n^k$	-	polynomiell
$2^n$	Alle Teilmengen untersuchen	exponentiell
$n!$	Alle Permutationen untersuchen	faktoriell

## Best, Worst und Average Case

Bei der Analyse von Algorithmen wird zwischen verschiedenen Fällen unterschieden. So dauert beispielsweise das Durchsuchen einer Liste (Marcos Suche) nach einem bestimmten Element, abhängig von dessen Position, unterschiedlich lange.

Befindet sich das Element gleich am Anfang, dann haben wir Glück, wir sprechen in diesem Fall von einer **best-case**-Eingabe. Im schlechtesten Fall, dem **worst-case**, steht es ganz am Ende und wir müssen die ganze Liste durchlaufen, bevor es gefunden wird. Im Mittel müssen ungefähr die Hälfte der Elemente angesehen werden, das ist der **average-case**.

Wenn wir also die Laufzeit im worst-case eines Algorithmus ermitteln wollen, gehen wir immer von der schlechtesten Ausgangsbedingung aus.

## Landau-Symbolik – $\Omega$ -Notation

Mit der  $\Omega$ -Notation geben wir eine untere Schranke an. Also analog zur  $O$ -Notation:

Eine Funktion  $f(n)$  hat als untere Schranke  $g(n)$ , wir schreiben dann  $f(n) \in \Omega(g(n))$ , wenn es eine positive Konstante  $n_0 \in \mathbb{N}$  gibt, so dass alle Werte von  $f(n)$  für alle  $n \geq n_0$  immer größer/gleich  $c \cdot g(n)$  sind.

Wir schreiben dafür kurz:

$$f(n) \in \Omega(g(n)): \exists n_0 \in \mathbb{N}, \exists c \in \mathbb{R}^+, \forall n \geq n_0: f(n) \geq c \cdot g(n)$$

Sollte gelten, dass  $f(n) \in O(g(n))$  und  $g(n) \in O(f(n))$  dann haben beide die gleiche Komplexität.

$$\Rightarrow f(n) \in \Theta(g(n))$$

Wir schreiben dafür kurz:

$$f(n) \in \Omega(g(n)): \exists n_0 \in \mathbb{N}, \exists c \in \mathbb{R}^+, \forall n \geq n_0: f(n) \geq c \cdot g(n)$$

## Landau-Symbolik – $\Theta$ -Notation

Sollte gelten, dass  $f(n) \in O(g(n))$  und  $g(n) \in O(f(n))$  dann haben beide die gleiche Komplexität.

$$\Rightarrow f(n) \in \Theta(g(n))$$

Wir schreiben dafür kurz:

$$f(n) \in \Theta(g(n)): \exists n_0 \in \mathbb{N}, \exists c_1, c_2 \in \mathbb{R}^+, \forall n \geq n_0: c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

## Landau-Symbolik – starke obere und untere Schranken

Eine **starke obere Schranke** können wir auch angeben über:

$$f(n) \in o(g(n)) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

In Mengenschreibweise:

$$o(g(n)) = \{f(n) \mid \forall c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : c \cdot g(n) \geq f(n)\}$$

Eine **starke untere Schranke** können wir auch angeben über:

$$f(n) \in \omega(g(n)) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$$

In Mengenschreibweise:

$$\omega(g(n)) = \{f(n) \mid \forall c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : c \cdot g(n) \leq f(n)\}$$

## Eselsbrücken

$$\begin{array}{ll} f(n) \in O(g(n)) & \leq \\ f(n) \in \Omega(g(n)) & \geq \\ f(n) \in \Theta(g(n)) & = \\ f(n) \in o(g(n)) & < \\ f(n) \in \omega(g(n)) & > \end{array}$$

## Definition über Grenzwerte von Quotientenfolgen

$$g \in O(f) \iff \overline{\lim}_{n \rightarrow \infty} \frac{g(n)}{f(n)} = c \iff f \in \Omega(g)$$

$$g \in o(f) \iff \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0 \iff f \in \omega(g)$$

$$g \in O(f) \wedge g \in \Omega(f) \iff g \in \Theta(f)$$

## Umgang mit Polynomen

Den praktischen Nutzen der Grenzwertregeln sehen wir am Beispiel des asymptotischen Wachstums von Polynomen.

Angenommen, wir haben als Laufzeit das folgende Polynom erhalten:

$$A(n): \alpha_k n^k + \alpha_{k-1} n^{k-1} + \dots + \alpha_1 n^1 + \alpha_0$$

Wir fordern, dass  $\alpha_k > 0$ , so entspricht  $k$  dem Grad des Polynoms  $A(n)$  und es gilt:

$$\lim_{n \rightarrow \infty} \frac{A(n)}{n^k} = \lim_{n \rightarrow \infty} (\alpha_k + \frac{\alpha_{k-1}}{n} + \dots + \frac{\alpha_1}{n^{k-1}} + \frac{\alpha_0}{n^k}) = \alpha_k$$

Wegen  $\alpha_k > 0$  folgt der Satz:

**Sei  $p(n)$  ein Polynom vom Grad  $k$ , dann gilt  $p(n) \in \Theta(n^k)$ .**

Wir erinnern uns an dieses Beispiel:

$$A(n): 3n^4 + 5n^3 + 7\log(n)$$

## Ein paar wichtige Regeln I

1. *Multiplikation mit Konstanten:* Für alle  $k > 0$  gilt:

$$k \cdot f(n) \in \Theta(f(n))$$

Das ist leicht mit dem Quotientenkriterium zu beweisen.

2. *Summen und Produkte:* Es gilt:

$$f_1(n) \in O(g_1(n)) \wedge f_2(n) \in O(g_2(n)) \Rightarrow f_1(n) + f_2(n) \in O(g_1(n) + g_2(n))$$

Das Gleiche gilt für Multiplikation statt Addition und für  $\Omega$ ,  $\omega$  und  $o$ . Es lässt sich ebenfalls mit dem Quotientenkriterium nachweisen.

3. *Potenzen:* Größere Potenzen sind starke obere Schranken von kleineren Potenzen:

$$a < b \Rightarrow n^a \in o(n^b)$$

Auch das folgt aus dem Quotientenkriterium.

## Ein paar wichtige Regeln II

4. *Polynome*: Es gilt:

$$(n + a)^b \in \Theta(n^b)$$

Das ist etwas überraschend. Es lässt sich aber überlegen, dass die Ungleichung gilt, wenn  $c = 2^b$  und  $n_0 = |a|$  für die obere Schranke und  $c = 2^{-b}$  und  $n_0 = 2 \cdot |a|$  für die untere Schranke gewählt werden. Aus dieser Regel folgt, dass in Polynomen das Wachstum immer durch die größte Potenz bestimmt wird.

5. *Logarithmen*: Alle Logarithmen (mit einer Basis  $> 1$ ) unterscheiden sich nur um eine Konstante, daher gilt:

$$\log_a n \in \Theta(\log_b n)$$

Das folgt direkt aus den Umwandlungsregeln für Logarithmen.

6. *Logarithmen gegen Potenzen*: Potenzen von Logarithmen wachsen langsamer als normale Potenzen:

$$(\log n)^a \in o(n^b)$$

Das lässt sich mit vollständiger Induktion beweisen.

## Ein paar wichtige Regeln III

7. *Potenzen gegen Exponentialfunktionen:* Potenzen wachsen immer langsamer als Exponentialfunktionen:

$$n^a \in o(b^n)$$

Auch das lässt sich mit vollständiger Induktion beweisen.

8. *Fakultäten:* Fakultäten lassen sich asymptotisch mit der Stirling Formel abschätzen. Es gilt dabei:

$$\log n! \in \Theta(n \log n)$$

Das ist leicht zu sehen, wenn  $n!$  durch  $n^n$  abgeschätzt wird.