

---

# **Herzlich Willkommen zur Programmkonstruktion**

---

# Am Anfang sagt man: „Hallo!“

```
public class Hello {  
    public static void main (String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

```
$ javac Hello.java  
$ java Hello  
Hello World!
```

---

# So schaut echter Java-Code aus

```
public class UnbekannteZahl {
    private int zahl;
    public UnbekannteZahl (int grenze) {
        zahl = (new Random()).nextInt() % grenze;
        if (zahl < 0) {
            zahl = zahl + grenze;
        }
    }
    public boolean gleich (int vergleichszahl) {
        return (zahl == vergleichszahl);
    }
    public boolean kleiner (int vergleichszahl) {
        return (zahl < vergleichszahl);
    }
}
```

---

# Zu Beginn ein kleines Spiel . . .

```
public class Zahlenraten {
    public static void main (String[] args) {
        UnbekannteZahl zuErraten = new UnbekannteZahl(100);
        Scanner sc = new Scanner(System.in);
        System.out.println("Errate eine Zahl zwischen 0 und 99:");
        while (sc.hasNext()) {
            if (sc.hasNextInt()) {
                int versuch = sc.nextInt();
                if (versuch < 0 || versuch > 99) {
                    System.out.println("Nur Zahlen von 0 bis 99!");
                }
                else if (zuErraten.gleich(versuch)) {
                    System.out.println("Gratulation! Zahl erraten!");
                    return;
                }
                else if (zuErraten.kleiner(versuch)) {
                    System.out.println("Gesuchte Zahl ist kleiner.");
                    ...
                }
            }
        }
    }
}
```

vollständiges Programm im Skriptum

---

## ... Zahlenraten für Glückspilze

```
$ javac Zahlenraten.java UnbekannteZahl.java
```

```
$ java Zahlenraten
```

```
Errate eine Zahl zwischen 0 und 99:
```

```
50
```

```
Die gesuchte Zahl ist kleiner.
```

```
Versuche es nocheinmal (Ende mit ^D):
```

```
25
```

```
Die gesuchte Zahl ist größer.
```

```
Versuche es nocheinmal (Ende mit ^D):
```

```
36
```

```
Die gesuchte Zahl ist größer.
```

```
Versuche es nocheinmal (Ende mit ^D):
```

```
43
```

```
Gratulation! Zahl erraten!
```

---

# Mehr als nur Programmieren:

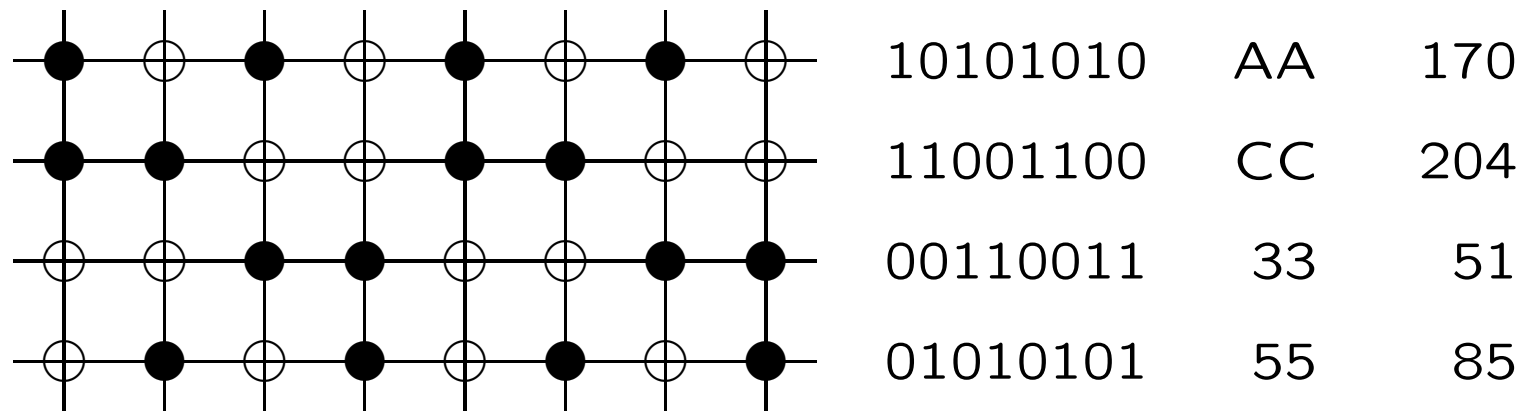
- Zusammenhänge im gesamten Umfeld:  
Maschinen, Programme, Softwareentwickler, Anwender
- grundlegende Konzepte und Aussagen der Informatik
- Techniken, Werkzeuge und Vorgehensweisen
- wie man beim Programmieren denkt
- handwerkliches Geschick und abstraktes Denkvermögen

---

# Digital & Binär

---

# Vom Webstuhl ins digitale Zeitalter



- analog = kontinuierlicher Wert = physikalische Größe, aber Fehler bei Verarbeitung und Speicherung unvermeidlich
- digital = diskreter Wert = Digitalisierungsfehler bei Messung, aber fehlerfrei verarbeit- und speicherbar



---

# Binäre Worte – Darstellungsformen

binär / oktäl / hexadezimal / dezimal / Zweierkomplement

|                        |                         |
|------------------------|-------------------------|
| 0000 = 00 = 0 = 00 = 0 | 1000 = 10 = 8 = 08 = -8 |
| 0001 = 01 = 1 = 01 = 1 | 1001 = 11 = 9 = 09 = -7 |
| 0010 = 02 = 2 = 02 = 2 | 1010 = 12 = A = 10 = -6 |
| 0011 = 03 = 3 = 03 = 3 | 1011 = 13 = B = 11 = -5 |
| 0100 = 04 = 4 = 04 = 4 | 1100 = 14 = C = 12 = -4 |
| 0101 = 05 = 5 = 05 = 5 | 1101 = 15 = D = 13 = -3 |
| 0110 = 06 = 6 = 06 = 6 | 1110 = 16 = E = 14 = -2 |
| 0111 = 07 = 7 = 07 = 7 | 1111 = 17 = F = 15 = -1 |

- ein  $n$ -Bit-Wort kann  $2^n$  Werte darstellen
- Dezimalzahlen:  $0 \dots 2^n - 1$  oder  $-2^{n-1} \dots 2^{n-1} - 1$

---

# Bytes

- früher: 1 Byte = Wortlänge für Darstellung von Zeichen
- heute: 1 Byte = 8 Bit (auch als Teil eines Wortes)
- noch immer: Größe in Byte  $\approx$  Textlänge in Zeichen

|      |                   |   |
|------|-------------------|---|
| 1 kB | = $2^{10}$ Byte = | 1.024 Byte $\approx 10^3$ Byte                |
| 1 MB | = $2^{20}$ Byte = | 1.048.576 Byte $\approx 10^6$ Byte            |
| 1 GB | = $2^{30}$ Byte = | 1.073.741.824 Byte $\approx 10^9$ Byte        |
| 1 TB | = $2^{40}$ Byte = | 1.099.511.627.776 Byte $\approx 10^{12}$ Byte |

---

# Bit-Operationen

- Logische Bit-Operationen:

|      |                    |                    |                    |                    |
|------|--------------------|--------------------|--------------------|--------------------|
| AND: | $00 \rightarrow 0$ | $01 \rightarrow 0$ | $10 \rightarrow 0$ | $11 \rightarrow 1$ |
| OR:  | $00 \rightarrow 0$ | $01 \rightarrow 1$ | $10 \rightarrow 1$ | $11 \rightarrow 1$ |
| XOR: | $00 \rightarrow 0$ | $01 \rightarrow 1$ | $10 \rightarrow 1$ | $11 \rightarrow 0$ |
| NOT: | $0 \rightarrow 1$  | $1 \rightarrow 0$  |                    |                    |

- Bit-Multiplikation entspricht UND:

$$0 \cdot 0 = 0 \quad 0 \cdot 1 = 0 \quad 1 \cdot 0 = 0 \quad 1 \cdot 1 = 1$$

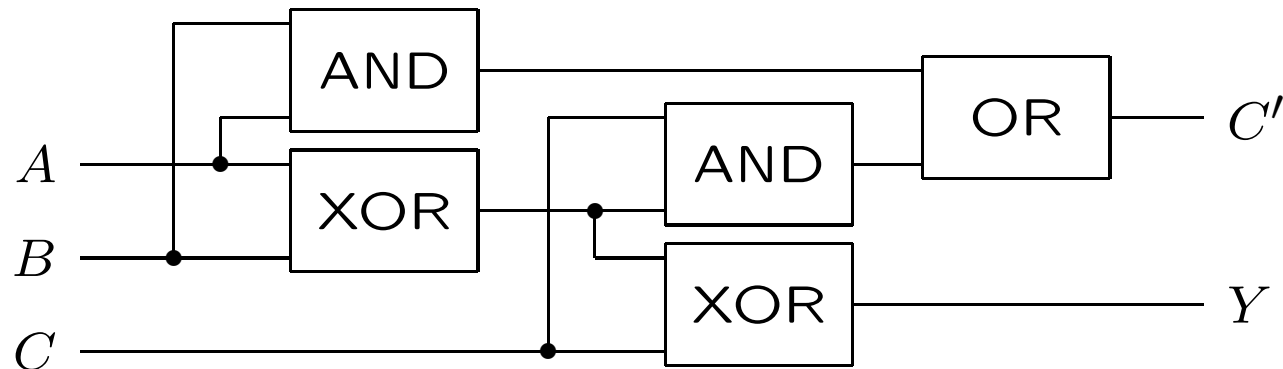
- Bit-Addition durch XOR, Übertrag durch AND:

$$0 + 0 = 00 \quad 0 + 1 = 01 \quad 1 + 0 = 01 \quad 1 + 1 = 10$$

---

# Logische Schaltung

Beispiel: Volladdierer



- Auch komplexe Schaltungen über einfache Gatter realisiert
- Gatterlaufzeiten ca. 0,1 ns bis über 100 ns
- Quelle und Ziel ist Speicher, Berechnung im fixen Takt

---

# VHDL-Programm für Hardware

```
library IEEE;                -- aus der Bibliothek IEEE:
use IEEE.std_logic_1164.all; -- Form der Bit-Darstellung

entity ANDGATE is            -- ein ANDGATE hat
  port (IN1 : in std_logic;  -- zwei Eingaenge (IN1, IN2)
        IN2 : in std_logic;  -- und einen Ausgang (OUT1)
        OUT1: out std_logic); -- als einfache Bits
end ANDGATE;

architecture RTL of ANDGATE is
begin
  OUT1 <= IN1 and IN2;       -- Berechnung des Ergebnisses
end RTL;
```

---

# Programmierbare Maschinen

- Computer sind einfach nur große logische Schaltungen
- mehr dazu am Donnerstag ...