UNIVERSITÄT
LEIPZIG

Einführung in die Objekt-Orientierte Modellierung und Programmierung

Wintersemester 2025/2026

Dirk Zeckzer

Institut für Informatik

Teil III

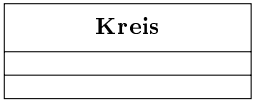
Klassen und Instanzen

Klasse, Instanz, Objekt

- **Klasse**
  - Abstrakt
  - Gruppe von Objekten
  - Festlegung der **Typen** (Eigenschaften) aller Attribute
- **Instanz**, **Objekt**
  - Konkret
  - Einzelnes Objekt
  - Festlegung der **Werte** (Wertebelegung) aller Attribute

Modellierung: UML – Java

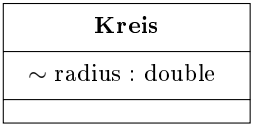
Name einer Klasse:




```
1  package eoomp;
2
3  _____ class Kreis {
4    // Block
5  }
```

Modellierung: UML – Java

Attribute einer Klasse:




```
1     _____ double radius;
```

Java: Primitive Datentypen

Datentyp	Wertebereich	Beispiele	bit
boolean	*true/false*		1
char	16 bit Unicode Zeichen (0x0000 - 0xFFFF)	`'a'`, `'1'`, `'Y'`	16

Java: Primitive Datentypen

Datentyp	Wertebereich	Beispiele	bit
`byte`	$-128 \ldots 127$ $(-2^7 \ldots 2^7 - 1)$	`-17, 0, 120, 127+1`	8
`short`	$-32.768 \ldots 32.767$ $(-2^{15} \ldots 2^{15} - 1)$		16
int	$-2.147.483.648 \ldots 2.147.483.647$ $(-2^{31} \ldots 2^{31} - 1)$		32
`long`	$\sim -9 \cdot 10^{18} \ldots \sim 9 \cdot 10^{18}$ $(-2^{63} \ldots 2^{63} - 1)$	`133L, -107l`	64

Java: Primitive Datentypen

Datentyp	Wertebereich	Beispiele	bit
`float`	$\sim \pm 1{,}402\mathrm{E}{-}45f$ $\ldots \sim \pm 3{,}4028\mathrm{E}{+}38f$	`-1.0f`	32
double	$\sim \pm 4{,}940\mathrm{E}{-}324$ $\ldots \sim \pm 1{,}79\mathrm{E}{+}308$	`1.0, -3.1, 1.7E05, 1.8E-7`	64

Java: Speicherverwaltung

Dezimalsystem:
Basis: 10
Ziffern: $0-9$
Zahlen:

$$5237 = 5 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 10^0$$

Allgemein:

$$\sum_{i=0}^{n} a_i \cdot 10^i \;=\; a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \ldots + a_2 \cdot 10^2 + a_1 \cdot 10^1 + a_0 \cdot 10^0$$

$$a_i \in \{0, \ldots, 9\}$$

Java: Speicherverwaltung

Binärsystem:
Basis: 2
Ziffern: $0 - 1$
Zahlen:

$$0100\,1011_2 = 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Allgemein:

$$\sum_{i=0}^{n} b_i \cdot 2^i \;=\; b_n \cdot 2^n + b_{n-1} \cdot 2^{n-1} + \ldots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

$$b_i \in \{0, 1\}$$

Java: Speicherverwaltung

Binärsystem $\rightarrow$ Dezimalsystem

$$
\begin{aligned}
0100\,1011_2 &= 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\
&= 64 + 8 + 2 + 1 \\
&= 75_{10}
\end{aligned}
$$

$$2^{10} = 1024 \approx 10^3$$

Java: Speicherverwaltung

- 1 **Bit**: 0 oder 1
- 1 **Byte** $\cong$ 8 Bit
  - Kleinste (positive) darstellbare Zahl $0000\,0000_2 = 0_{10}$
  - Größte (positive) darstellbare Zahl $1111\,1111_2 = 255_{10}$
  - $2^8 = 256$ verschiedene Zahlen/Zustände/Adressen

Länge	Zahlen/Zustände/Adressen	
8 bit	2^8	256
16 bit	2^{16}	65536
32 bit	2^{32}	$> 4 \cdot 10^9$
64 bit	2^{64}	$> 16 \cdot 10^{18}$

Java: Speicherverwaltung

Hexadezimalsystem:
Basis: 16
Ziffern: $0-9, A-F$
Zahlen:

$$4B_{16} = 4 \cdot 16^1 + 11 \cdot 16^0$$

Allgemein:

$$\sum_{i=0}^{n} b_i \cdot 16^i = b_n \cdot 16^n + b_{n-1} \cdot 16^{n-1} + \ldots + b_2 \cdot 16^2 + b_1 \cdot 16^1 + b_0 \cdot 16^0$$

$$b_i \in \{0-9, A-F\}$$

Java: Speicherverwaltung

Binär	Dezimal	Hexadezimal
0000_2	0	0_{16}
0001_2	1	1_{16}
0010_2	2	2_{16}
0011_2	3	3_{16}
0100_2	4	4_{16}
0101_2	5	5_{16}
0110_2	6	6_{16}
0111_2	7	7_{16}

Binär	Dezimal	Hexadezimal
1000_2	8	8_{16}
1001_2	9	9_{16}
1010_2	10	A_{16}
1011_2	11	B_{16}
1100_2	12	C_{16}
1101_2	13	D_{16}
1110_2	14	E_{16}
1111_2	15	F_{16}

$$0100\,1011_2 = 4B_{16}$$

$$\begin{aligned} 4B_{16} &= 4 \cdot 16^1 + 11 \cdot 16^0 \\ &= 64 + 11 \\ &= 75_{10} \end{aligned}$$

ASCII

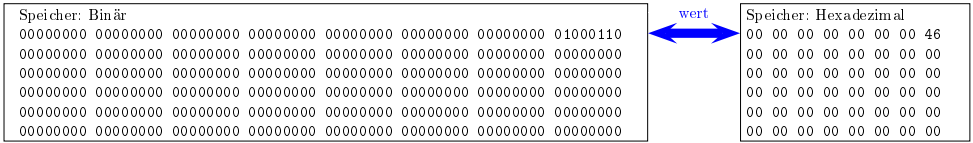
- ASCII: **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- 7-Bit-Zeichenkodierung (128 Zeichen)
- 95 druckbare Zeichen
- 33 nicht druckbare Zeichen, zum Beispiel
  - `CR`: **carriage return** - springe zurück zum Zeilenanfang
  - `LF`: **line feed** - springe eine Zeile nach unten
  - `EOL`: **end of line**
  - `EOF`: **end of file**
  - `␣`: das Leerzeichen

ASCII

Code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Java: Speicherverwaltung

- Daten im Speicher im Binärformat
- Daten als Java primitiver Datentyp
  - `int wert` → 70
  - `char wert` → 'F' (ASCII)
  - ...



Java: Array

Arrays:

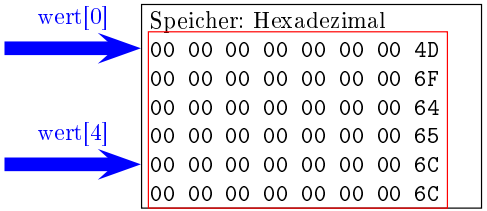
```
1  int[] werte = new int[4];
2  werte[0] = 1;
3  werte[1] = 2;
4  werte[2] = 3;
5  werte[3] = 4;
```

`werte.length` : Länge des Arrays

Java: Speicherverwaltung

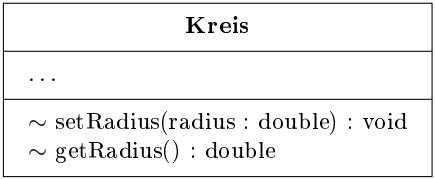
Arrays

```
1    char[] wert
2      = new char[6];
3    wert[0] = 'M';
4    wert[1] = 'o';
5    wert[2] = 'd';
6    wert[3] = 'e';
7    wert[4] = 'l';
8    wert[5] = 'l';
```



Modellierung: UML – Java

Methoden einer Klasse:




```
1  _____ void setRadius(
2    double radius
3  ) {
4    this.radius = radius;
5  }
6
7  _____ double getRadius()
       {
8    return radius;
9  }
```

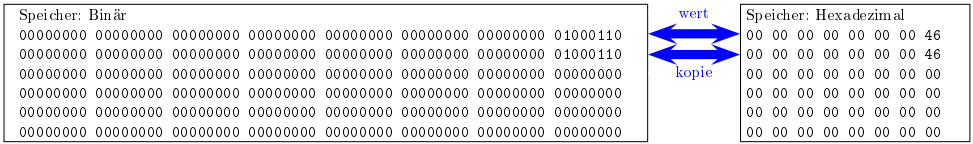
Modellierung: UML – Java

Konstruktor einer Klasse:


```
1  _____ Kreis(
2    double radius
3  ) {
4    this.radius = radius;
5  }
```

Java: Speicherverwaltung

- Kopieren von Werten
  - `int wert` $\rightarrow$ 70
  - `int kopie = wert;`



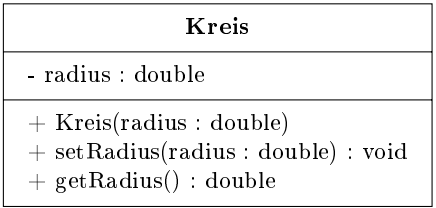
Modellierung: UML – Java

Sichtbarkeit von Attributen und Methoden einer Klasse A in einer anderen Klasse:

UML	Java	Bemerkung
$+$	`public`	in allen Klassen
$-$	`private`	nur innerhalb der Klasse A
$\#$	`protected`	in allen Klassen, die Klasse A erweitern und in allen Klassen im selben *package*
$\sim$		in allen Klassen im selben *package*
$/$		*derived*, keine Entsprechung in Java

Modellierung: UML – Java

Sichtbarkeit einer Klasse:




```java
package eoomp;

public class Kreis {

  private double radius;

  public Kreis(
    double radius
  ) {
    this.radius = radius;
  }

  public void setRadius(
    double radius
  ) {
    this.radius = radius;
  }

  public double getRadius() {
    return radius;
  }
}
```

Java: Class ⇔ Instance

new

```
1   Kreis kreis1 = new Kreis(1.0);
2   kreis1.setRadius(5.0);
3   double radius1 = kreis1.getRadius();
4
5   Kreis kreis2 = new Kreis(7.0);
6   double radius2 = kreis2.getRadius();
```

main

```
1   public static void main (
2     String[] args
3   ) {
4     ...
5   }
```

main in einer eigenen Klasse

```
1   package eoomp;
2
3   public class KreisMain
4   {
5     public static void main (
6       String[] args
7     ) {
8       Kreis kreis1 = new Kreis(1.0);
9       kreis1.setRadius(5.0);
10      double radius1 = kreis1.getRadius();
11
12      Kreis kreis2 = new Kreis(7.0);
13      double radius2 = kreis2.getRadius();
14    }
15  }
```

Signatur einer Methode

- Die **Signatur** einer Methode umfasst folgende Elemente
  1. Den Namen
  2. Die Typen und ihre Reihenfolge in der Parameterliste

- Weitere Elemente der Deklaration einer Methode:
  1. Die Sichtbarkeit (`public`, “package” (kein Schlüsselwort), `protected`, `private`)
  2. Ob die Methode `static` ist, oder nicht
  3. Ob die Methode `final` ist, oder nicht
  4. Der Typ der Rückgabe (erforderlich)

Konstruktoren

- Sind spezielle Methoden
- Signatur: umfasst die selben Element wie eine Methode
  1. Den Namen des Konstruktors. Dieser ist identisch mit dem Namen der Klasse
  2. Die Typen und ihre Reihenfolge in der Parameterliste
- Weitere Elemente sind
  - die Sichtbarkeit
  - ggf. `static`
  - ggf. `final`
- **Kein Rückgabetyp**
- **Rückgabetyp**:
  die Klasse selbst
- **Rückgabewert**:
  eine neue Instanz der Klasse
- Aufruf mit
  - `new`
  - Name der Klasse
    (Name des Konstruktors)
  - Parameterliste

Die Methode `main`

- Die Methode `main` hat immer folgende Form

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

- Es ist die Methode, welche beim Start eines Programmes aufgerufen wird.