

# Lektion 6

Methoden  
Rekursive Methoden

# Methoden

```

public static void main(String[] args)
{
    System.out.println("Was wollen Sie tun?");
    System.out.println("1: PI-Berechnung");
    System.out.println("2: Quadratwurzelberechnung");
    System.out.println("3: Betrag");
    //...
    Scanner scanner = new Scanner(System.in);
    int auswahl = scanner.nextInt();
    if (auswahl == 1)
    {
        //PI-Berechnung
        //verwendet Quadratwurzelberechnung
    }
    else if (auswahl == 2)
    {
        //Quadratwurzelberechnung
    }
    else if (auswahl == 3)
    {
        //Betrag
    }
    //...
}

```

Wenn wir mehrere Berechnungen in einem Programm unterbringen wollen, kann es schnell unübersichtlich werden.

Wir benötigen eine bessere Unterteilung in einzelne Programmteile!



Dies geschieht durch sog. „Methoden“.

# Methoden

- Eine Methode ähnelt einer mathematischen Funktion, sie nimmt ein (od. mehrere) Argument(e) entgegen und hat einen Rückgabewert.

$$f(x_0) = y$$

Input:  $x_0$ , Resultat:  $y$

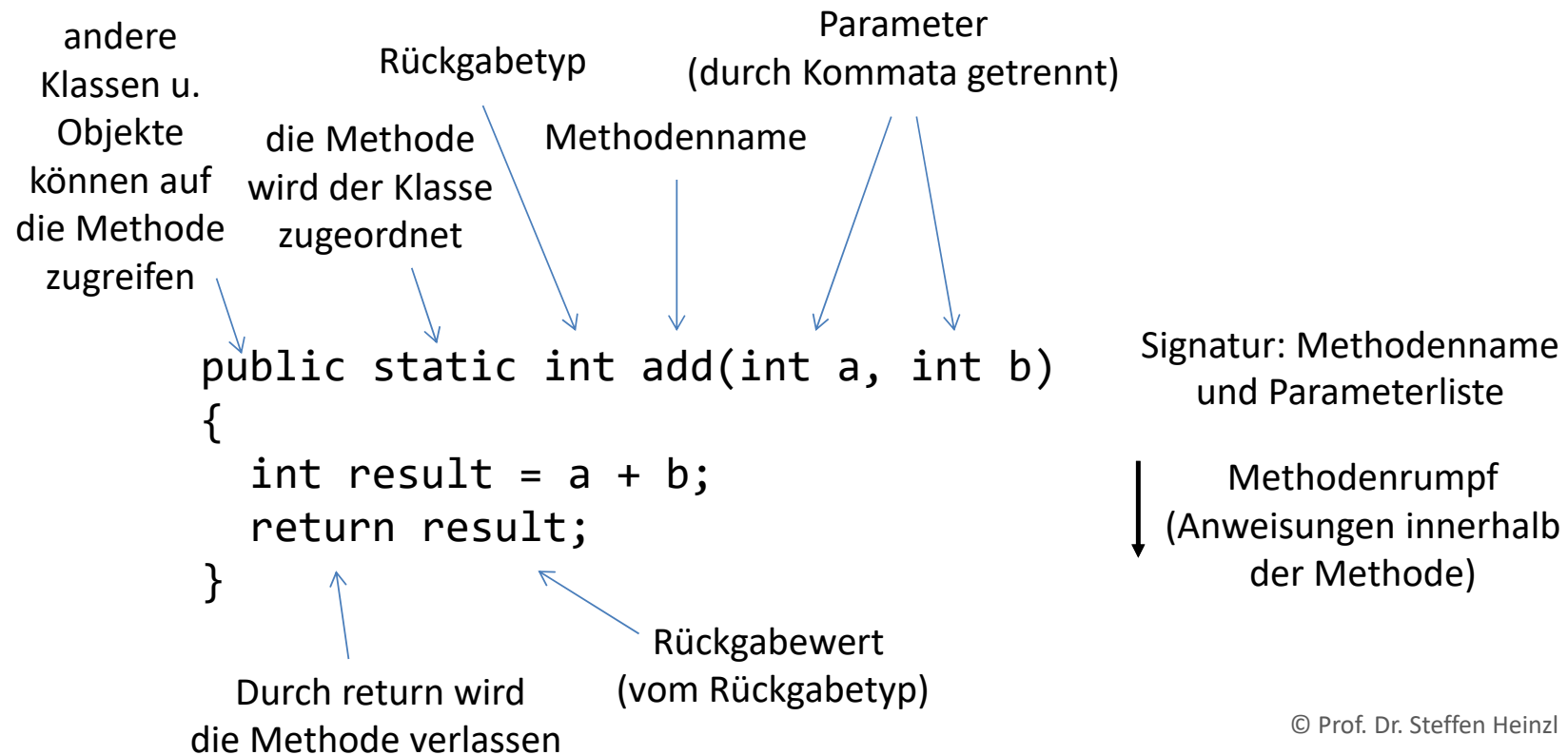
- Eine Methode kann also als eine Art Unterprogramm verwendet werden, die eine (od. mehrere) Eingaben verarbeitet und einen Wert zurückgibt.
- Durch solche Unterprogramme können Teilprobleme, die immer wieder gelöst werden müssen, gekapselt werden, z.B.:
  - Berechnung des verzinsten Kapitals nach  $n$  Jahren
  - Berechnung des Kosinus
  - Erkennung des Vorzeichens
  - Hinzufügen von Daten in eine Tabelle
  - Download einer Webseite
  - Generierung eines Ebooks
  - ...

## Welche Methoden kennen wir eigentlich schon?

Methoden	Rückgabewert	Parameter
System.out.println	void	String oder int oder double, etc.
Math.pow	double	double, double
string.charAt	char	int
scanner.nextInt	int	-
...		

# Methoden – Beispiel

- Eine Methode, die eine Addition zweier Ganzzahlen durchführt, könnte folgendermaßen aussehen:



```
public static int add(int a, int b)
{
    int result = a + b;
    return result;
}
```

Wie kann ich die Methode  
aufrufen?

Durch Schreiben von

`<Methodenname>(<Argument1>, <Argument2>, ...)`

Beispiel:

`add(5,7);`

oder:

`int ergebnis;  
ergebnis = add(5,7);`

# Methoden - return

- Mit return kann eine Methode auch vorzeitig verlassen werden.
- Beispiel: Eine Methode, die das Vorzeichen einer Zahl bestimmt:

```
public static int sign(double zahl)
{
    if (zahl < 0)
        return -1;
    else if (zahl > 0)
        return +1;
    else
        return 0;
}
```



# Methodenaufruf – Beispiel

```
public class MethodenBeispiel
{
    public static int sign(double x)
    {
        if (x < 0) return -1;
        else if (x > 0) return +1;
        else return 0;
    }
    public static void main(String[] args)
    {
        //Betrag berechnen
        int zahl = -5;
        zahl = zahl * sign(zahl);
        System.out.println(zahl);
    }
}
```

# Methodenaufruf – Beispiel

```
public class MethodenBeispiel
{
    public static int sign(double x)
    {
        if (x < 0) return -1;
        else if (x > 0) return +1;
        else return 0;
    }
    public static void main(String[] args)
    {
        //Betrag berechnen
        int zahl = -5;
        zahl = zahl * sign(zahl);
        System.out.println(zahl);
    }
}
```

wird zu -5 ausgewertet

wird zu -1 ausgewertet

-5 \* -1, also 5 wird in *zahl* geschrieben

# Methode - Rückgabetypen

- Als Rückgabetyper kommen jegliche Datentypen in Frage.

void-Rückgabetyper bedeutet,  
dass die Methode keinen Wert zurückgibt.

```
public static void printMenue()  
{  
    System.out.println("*****");  
    System.out.println("* 1: Werte einlesen  *");  
    System.out.println("* 2: Formel berechnen  *");  
    System.out.println("*****");  
    return;  
}
```

Die Methode kann mit einem  
return (auch vorzeitig) verlassen werden

...

//Aufruf:

```
printMenue();
```

# Methodennamen

- Methodennamen beginnen wie Variablennamen mit einem Kleinbuchstaben und verwenden **Camel Case**.
- Methoden werden häufig nach Tätigkeiten benannt:
  - gibMenueAus, erstelleDokument, ...
- Ausnahme sind bspw. mathematische Methoden: Hier wird der bereits bekannte Name aus der Mathematik genommen.
  - sin, cos, ...

Warum ist der Einsatz von  
Methoden sinnvoll?

z. B. Wiederverwendung von Code

Wir haben bspw. für mehrere Aufgaben  
bereits Potenzrechnung verwendet.

Aber: Wie schreibt man eine Methode, die  
die Potenz berechnet?

## Erste Überlegung:



Welchen Namen hat die Methode  
und welches Ergebnis?

Der **Name** sollte das kurz bezeichnen, was  
die Methode macht, z. B. **potenz**

Das Ergebnis der Potenz ist eine **Zahl**.

➤ Die Methode gibt also eine Zahl zurück.

```
public static int potenz()
```

## Zweite Überlegung:



Welche Argumente verarbeitet die Methode?  
(Was gebe ich in die Methode hinein?)

Zur Berechnung der Potenz wird Basis und  
Exponent benötigt.

➤ Die Methode hat **zwei Parameter**.

```
public static int potenz(int basis, int exponent)
{
}
}
```

Dritte Überlegung:  
Wie berechne ich die Potenz?

$$x^n = \underbrace{x \cdot x \cdot x \dots \cdot x}_{n\text{-mal}}$$



➤ Schleife



## Wie berechne ich die Potenz?

```
int x = 3;
int n = 4;
int ergebnis = 1; //neutrales Element der Multiplikation

for (int i = 1; i <= n; i++)
{
    ergebnis = ergebnis * x;
}
System.out.println(ergebnis);
```

$$1 \cdot x \cdot x \cdot x \cdot x$$

# Zusammenführung von Methodenkopf und Implementierung



```
public static int potenz(int basis, int exponent)
{
    int x = 3;
    int n = 4;
    int ergebnis = 1; //neutrales Element der Multiplikation

    for (int i = 1; i <= n; i++)
    {
        ergebnis = ergebnis * x;
    }
    System.out.println(ergebnis);
}
```

# Zusammenführung von Methodenkopf und Implementierung



```
public static int potenz(int basis, int exponent)
{
    int x = 3;
    int n = 4;
    int ergebnis = 1; //neutrales Element der Multiplikation

    for (int i = 1; i <= n; i++)
    {
        ergebnis = ergebnis * x;
    }
    System.out.println(ergebnis);
}
```

wir bekommen von außen die Werte  
für **basis** und **exponent**, rechnen  
aber momentan mit **x** und **n**

- wir müssen die Werte in der  
Methode nicht selbst  
deklarieren

# Zusammenführung von Methodenkopf und Implementierung



```
public static int potenz(int basis, int exponent)
{

    int ergebnis = 1; //neutrales Element der Multiplikation

    for (int i = 1; i <= n; i++)
    {
        ergebnis = ergebnis * x;
    }
    System.out.println(ergebnis);
}
```

Die bisherigen Werte x und n  
müssen wir mit basis und exponent  
ersetzen.

# Zusammenführung von Methodenkopf und Implementierung



```
public static int potenz(int basis, int exponent)
{

    int ergebnis = 1; //neutrales Element der Multiplikation

    for (int i = 1; i <= exponent; i++)
    {
        ergebnis = ergebnis * x;
    }
    System.out.println(ergebnis);
}
```

die bisherigen Werte x und n  
müssen wir mit basis und exponent  
ersetzen.

# Zusammenführung von Methodenkopf und Implementierung



```
public static int potenz(int basis, int exponent)
{

    int ergebnis = 1; //neutrales Element der Multiplikation

    for (int i = 1; i <= exponent; i++)
    {
        ergebnis = ergebnis * basis;
    }
    System.out.println(ergebnis);
}
```

die bisherigen Werte x und n  
müssen wir mit basis und exponent  
ersetzen.

## Zusammenführung von Methodenkopf und Implementierung

```
public static int potenz(int basis, int exponent)
{

    int ergebnis = 1; //neutrales Element der Multiplikation

    for (int i = 1; i <= exponent; i++)
    {
        ergebnis = ergebnis * basis;
    }
    return ergebnis;
}
```

Die Methode muss einen Rückgabewert vom Typ int haben.

# Wie teste ich die Methode?

```
public class PotenzBeispiel
{
    public static int potenz(int basis, int exponent)
    {
        int ergebnis = 1; //neutrales Element der Multiplikation
        for (int i = 1; i <= exponent; i++)
        {
            ergebnis = ergebnis * basis;
        }
        return ergebnis;
    }

    public static void main(String[] args)
    {
        int ergebnis = potenz(3, 4);
        System.out.println(ergebnis);
    }
}
```



Kann die Methode noch sinnvoll  
verallgemeinert werden?

```
public static int potenz(int basis, int exponent)
{
    int ergebnis = 1; //neutrales Element der Multiplikation
    for (int i = 1; i <= exponent; i++)
    {
        ergebnis = ergebnis * basis;
    }
    return ergebnis;
}
```

Wird ein Wert verwendet, den man variabel  
machen könnte?

Können Datentypen ausgeweitet werden?

## Kann die Methode noch sinnvoll verallgemeinert werden?

```
public static int potenz(int basis, int exponent)
{
    int ergebnis = 1; //neutrales Element der Multiplikation
    for (int i = 1; i <= exponent; i++)
    {
        ergebnis = ergebnis * basis;
    }
    return ergebnis;
}
```



Ausweitung von Datentypen

```
public static double potenz(double basis, int exponent)
{
    double ergebnis = 1; //neutrales Element der Multiplikation
    for (int i = 1; i <= exponent; i++)
    {
        ergebnis = ergebnis * basis;
    }
    return ergebnis;
}
```

# Nochmal Sichtbarkeit von Variablen

```
public class PotenzBeispiel
{
    public static int potenz(int basis, int exponent)
    {
        int ergebnis = 1; //neutrales Element der Multiplikation
        for (int i = 1; i <= exponent; i++)
        {
            ergebnis = ergebnis * basis;
        }
        return ergebnis;
    }
}

public static void main(String[] args)
{
    int ergebnis = potenz(3, 4);
    System.out.println(ergebnis);
}
}
```

basis und exponent sind sichtbar

# Nochmal Sichtbarkeit von Variablen

```
public class PotenzBeispiel
```

```
{
```

```
    public static int potenz(int basis, int exponent)
```

```
    {
```

```
        int ergebnis = 1; //neutrales Element der Multiplikation
```

```
        for (int i = 1; i <= exponent; i++)
```

```
        {
```

```
            ergebnis = ergebnis * basis;
```

```
        }
```

```
        return ergebnis;
```

```
    }
```

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

```
    {
```

```
        int ergebnis = potenz(3, 4);
```

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

```
    }
```

```
}
```

ergebnis ist sichtbar



Zwei unterschiedliche  
Variablen (u. Speicherplätze)  
trotz gleichen Namens

ergebnis ist sichtbar

# Methodenbeispiel - Betrag

```
public class MethodenBeispiel
{
    public static double absoluteValue(double x)
    {
        if (x < 0) return -x;
        else return x;
    }
    public static void main(String[] args)
    {
        System.out.println(absoluteValue(-5));
        System.out.println(absoluteValue(5));
        System.out.println(absoluteValue(0));
    }
}
```

# Methode verwendet Methode

```
public class MethodenBeispiel2
{
    public static double absoluteValue(double x)
    {
        return x*sign(x);
    }
    public static int sign(double zahl)
    {
        if (zahl < 0) return -1;
        else if (zahl > 0) return +1;
        else return 0;
    }
    public static void main(String[] args)
    {
        System.out.println(absoluteValue(-5));
    }
}
```

Methode absoluteValue  
verwendet Methode sign zur  
Bestimmung des Betrags

# Einfache 2D-Grafik

# Zum Ausprobieren: Graphik

- Java kann mehr als nur die Kommandozeile:

```
import java.applet.Applet;
import java.awt.*;

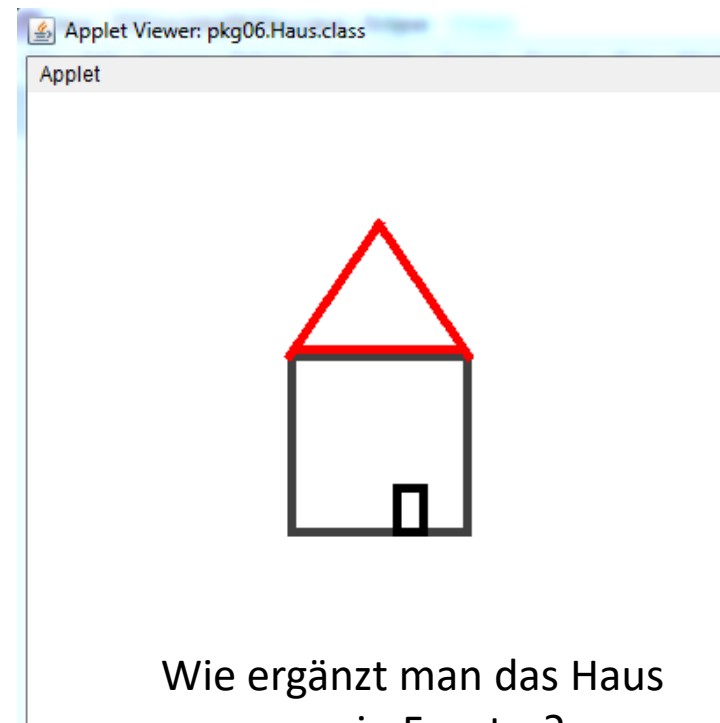
public class Haus extends Applet {
    public void paint (Graphics g) {
        Graphics2D g2D = (Graphics2D) g;
        //Haus
        g2D.setColor(Color.DARK_GRAY);
        g2D.setStroke(new BasicStroke(5)); //Pinselfbreite 5
        g2D.drawRect(150, 150, 100, 100);
        //Dach
        g2D.setColor(Color.RED);
        g2D.drawLine(149, 150, 200, 75);
        g2D.drawLine(251, 150, 200, 75);
        g2D.drawLine(154, 146, 247, 146);
        //Tür
        g2D.setColor(Color.BLACK);
        g2D.drawRect(210, 225, 15, 25);
    }
}
```



# Zum Ausprobieren: Graphik

- Java kann mehr als nur Programme auf der Kommandozeile:

```
import java.applet.Applet;  
import java.awt.*;  
  
public class Haus extends Applet {  
    public void paint (Graphics g) {  
        Graphics2D g2D = (Graphics2D) g;  
        //Haus  
        g2D.setColor(Color.DARK_GRAY);  
        g2D.setStroke(new BasicStroke(5)); //Pinselfbreite 5  
        g2D.drawRect(150, 150, 100, 100);  
        //Dach  
        g2D.setColor(Color.RED);  
        g2D.drawLine(149, 150, 200, 75);  
        g2D.drawLine(251, 150, 200, 75);  
        g2D.drawLine(154, 146, 247, 146);  
        //Tür  
        g2D.setColor(Color.BLACK);  
        g2D.drawRect(210, 225, 15, 25);  
    }  
}
```



Wie ergänzt man das Haus  
um ein Fenster?

# Zum Ausprobieren: Graphik

- Java kann mehr als nur Programme auf der Kommandozeile:

```
import java.applet.Applet;
import java.awt.*;

public class Haus extends Applet {
    public void paint (Graphics g) {
        Graphics2D g2D = (Graphics2D) g;
        //Haus
        g2D.setColor(Color.DARK_GRAY);
        g2D.setStroke(new BasicStroke(5)); //Pinselfbreite 5
        g2D.drawRect(150, 150, 100, 100);
        //Dach
        g2D.setColor(Color.RED);
        g2D.drawLine(149, 150, 200, 75);
        g2D.drawLine(251, 150, 200, 75);
        g2D.drawLine(154, 146, 247, 146);
        //Tür
        g2D.setColor(Color.BLACK);
        g2D.drawRect(210, 225, 15, 25);
    }
}
```



Applet ist seit Java 9  
veraltet und wurde  
entfernt...

Wie ergänzt man das Haus  
um ein Fenster?

# Zum Ausprobieren: Graphik

```
import java.swing.*;  
import java.awt.*;
```

```
public class Haus  
{  
    public Haus()  
    {  
        JFrame frame = new JFrame();  
        JComponent panel = new JComponent()  
        {  
            @Override  
            public void paintComponent(Graphics g)  
            {  
                super.paintComponent(g);  
                zeichneHaus((Graphics2D) g);  
            }  
        };  
        frame.add(panel);  
        frame.setSize(1024, 768);  
        frame.setVisible(true);  
    }  
    //rechts geht weiter ->
```

```
        public void zeichneHaus(Graphics2D g2D)  
        {  
            // Haus  
            g2D.setColor(Color.DARK_GRAY);  
            g2D.setStroke(new BasicStroke(5)); // Pinselbreite 5  
            g2D.drawRect(150, 150, 100, 100);  
            // Dach  
            g2D.setColor(Color.RED);  
            g2D.drawLine(149, 150, 200, 75);  
            g2D.drawLine(251, 150, 200, 75);  
            g2D.drawLine(154, 146, 247, 146);  
            // Tür  
            g2D.setColor(Color.BLACK);  
            g2D.drawRect(210, 225, 15, 25);  
        }  
  
        public static void main(String[] args)  
        {  
            new Haus();  
        }  
    } //end of class Haus
```

...daher müssen wir  
das Bsp. etwas  
komplexer gestalten

# Rekursion

# Aufgaben: Potenz, Fakultät

Sei  $x, y \in \mathbb{R}$  beliebig,  $n \in \mathbb{N}$  beliebig. Vereinfache:

1)  $x^3 \cdot x^2$

2)  $\frac{x^{n+1}}{x^{n-1}}, x \neq 0$

3)  $x^{3^2}$

4)  $(x^3)^2$

5)  $\frac{(n+1)!}{n!}$

6)  $\frac{(2n)!}{(2n+3)!}$

# Rekursive Methodenaufrufe

- Unter Rekursion versteht man, dass eine Methode sich selbst erneut aufruft.
  - Dabei wird in der Regel ein veränderter Parameter übergeben.
  - Die Aufrufe erfolgen solange, bis eine bestimmte Abbruchbedingung erfüllt ist.
- Rekursionen kann man anstelle von Schleifen verwenden.

# Fakultät – Induktive Definition

- Induktive (rekursive) Definition der Fakultät:

Definition:

Für  $n \in \mathbb{N}$  gilt:

$$0! := 1$$

$$(n + 1)! := (n + 1) \cdot n!$$

Beispiel:

$$5! = 5 \cdot 4!$$

$$= 5 \cdot 4 \cdot 3!$$

$$= 5 \cdot 4 \cdot 3 \cdot 2!$$

$$= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1!$$

$$= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \cdot 0!$$

$$= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1$$

weitere Beispiele:

$$n! = n \cdot (n - 1) \cdot (n - 2) \dots \cdot 3 \cdot 2 \cdot 1 \cdot 0!$$

# Fakultät – Rekursive Berechnung

Die Fakultät kann in einer rekursiven Methode berechnet werden:

```
public static long fak(int n)
{
    if (n==0) return 1;
    return n * fak(n-1);
}
```

Wenn n gleich 0 ist,  
wird die Methode verlassen  
und die Rekursion beendet.

$0! := 1$

$(n + 1)! := (n + 1) \cdot n!$



# Summe – rekursive Berechnung

- Analog zur Fakultät lässt sich auch die Summe in einer rekursiven Methode berechnen.
- Mathematisch:

$$\sum_{k=0}^n k = n + \sum_{k=0}^{n-1} k \quad , n > 0$$

- Als Methode:


```
public static int sum(int n)
{
    if (n == 0) return 0;
    return n + sum(n-1);
}
```

← Wird die Summe von der  
Zahl 0 gebildet, wird 0  
zurückgegeben.

## Rekursive Aufrufe – Beispiel

```
public static int sum(int n)
{
    if (n == 0) return 0;
    return n + sum(n-1);
}
```

```
public static void main(String[] args)
{
    System.out.println(sum(3));
}
```



# Rekursive Aufrufe – Beispiel

```
public static int sum(int n)
{
    if (3 == 0) return 0;
    return 3 + sum(3-1);
}

2 public static int sum(int n)
{
    if (2 == 0) return 0;
    return 2 + sum(2-1);
}

1 public static int sum(int n)
{
    if (1 == 0) return 0;
    return 1 + sum(1-1);
}

0 public static void main(String[] args)
{
    System.out.println(sum(3));
}
```

## Rekursive Aufrufe – Beispiel (ctd.)

```
public static int sum(int n)
{
    if (3 == 0) return 0;
    return 3 + sum(3-1);
}
    6

public static int sum(int n)
{
    if (2 == 0) return 0;
    return 2 + sum(2-1);
}
    3

public static int sum(int n)
{
    if (1 == 0) return 0;
    return 1 + sum(1-1);
}
    0

public static int sum(int n)
{
    if (0 == 0) return 0;
    return 0 + sum(0-1);
}

public static void main(String[] args)
{
    System.out.println(sum(3));
}
```

## Rekursive Aufrufe – Beispiel (alternativ)

```
public static int sum(int n)
{
    if (n == 0) return 0;
    return n + sum(n-1);
}
```

$n == 0$  ist falsch, daher return  $3 + \text{sum}(2)$ . Damit die Rückgabe erfolgen kann, muss zunächst  $\text{sum}(2)$  evaluiert werden.

$n == 0$  ist falsch, da  $n = 2$ . Daher return  $2 + \text{sum}(1)$ . Damit die Rückgabe erfolgen kann, muss zunächst  $\text{sum}(1)$  evaluiert werden.

$n == 0$  ist falsch, da  $n = 1$ . Daher return  $1 + \text{sum}(0)$ . Damit die Rückgabe erfolgen kann, muss zunächst  $\text{sum}(0)$  evaluiert werden.

$n == 0$  ist wahr, daher return 0;

Nun werden die Ausdrücke wieder rückwärts zusammengesetzt:

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

$\text{sum}(1) = 1 + \text{sum}(0) = 1$

$\text{sum}(2) = 2 + \text{sum}(1) = 2 + 1 = 3$

$\text{sum}(3) = 3 + \text{sum}(2) = 3 + 3 = 6$

Welche Lösung ist die richtige, um folgende induktive Definition mit einer rekursiven Methode in Java abzubilden?

$$\sum_{k=m}^n k := n + \sum_{k=m}^{n-1} k, n > m$$

$$\sum_{k=m}^n k := m, n = m$$

A:

```
public static int sum(int m, int n)
{
    if (n==m) return m;
    else return sum(m,n-1);
}
```

B:

```
public static int sum(int m, int n)
{
    if (n==m) return m;
    else return n + sum(n-1);
}
```

C:

```
public static int sum(int m, int n)
{
    if (n==m) return m;
    else return n + sum(m,n-1);
}
```

D:

```
public static int sum(int m, int n)
{
    if (n==0) return m;
    else return n + sum(m,n-1);
}
```

# Potenz – Induktive Definition

- Induktive (rekursive) Definition der Potenz:

Definition:

Für  $n \in \mathbb{N}, x \in \mathbb{R}$  gilt:

$$x^0 := 1$$

$$x^{n+1} := x^n \cdot x$$

Beispiele:

$$3^0 = 1$$

$$0^0 = 1$$

$$5^4 = 5^3 \cdot 5$$

$$= 5^2 \cdot 5 \cdot 5$$

$$= 5^1 \cdot 5 \cdot 5 \cdot 5$$

$$= 5^0 \cdot 5 \cdot 5 \cdot 5 \cdot 5$$

$$= 1 \cdot 5 \cdot 5 \cdot 5 \cdot 5 = 625$$

Wie kann die Potenz in einer  
rekursiven Methode  
berechnet werden?

Zur Übung!