

Javakurs 2010 – LE3

Methoden, Testen, Debuggen

Sebastian Dyroff, Mario Bodemann

23. März 2010



This work is licensed under the *Creative Commons Attribution-ShareAlike 3.0 License*.

created with \LaTeX -beamer

Agenda 2010

- 1 Methoden
- 2 Testen
- 3 Debuggen
- 4 Java-API

Feedback of the Day

Feedback

Feedback: Vorlesung

☒ vormittags ☐ nachmittags

☒ Montag ☐ Dienstag
☐ Mittwoch ☐ Donnerstag
☐ Freitag

Positiv

Lautstärke gut



Negativ

hellgrüne Schriftfarbe auf Folie = schlecht



0. Wiederholung

Wiederholung

Wiederholung

- Variablen und Zuweisungen

Wiederholung

- Variablen und Zuweisungen

```
1 int foo = 42;  
2 String text = "Hallo Welt!";  
3  
4 int bar;  
5 bar = 23;
```

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen

```
1  if ( heuteIstRasenmaehertag == true ) {  
2      System.out.println("Geh Rasen maehen!");  
3  } else {  
4      System.out.println("Faulenzen!");  
5  }
```

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen

```
1 System.out.println("Ich");  
2 for(int count=0; count<10; count++) {  
3     System.out.println("maehe");  
4 }
```

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen

```
1 System.out.println("Ich");
2 for(int count=0; count<10; count++) {
3     System.out.println("maehe");
4 }
```

```
1 System.out.println("Ich");
2 int count = 0;
3 while(count<10) {
4     System.out.println("maehe");
5     count++;
6 }
```

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen
- Arrays

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen
- Arrays

```
1  int [] grashalme = new int[10];  
2  grashalme[0] = 0;  
3  grashalme[1] = 0;  
4  grashalme[2] = 0;  
5  ...  
6  grashalme[9] = 0;
```


Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen
- Arrays

```
1  int [] grashalme = new int[10];  
2  grashalme[0] = 0;  
3  grashalme[1] = 0;  
4  grashalme[2] = 0;  
5  ...  
6  grashalme[9] = 0;
```

```
1  int [] grashalme = new int[10];  
2  for(int halmNr=0; halmNr<grashalme.length; halmNr++) {  
3      grashalme[halmNr] = 0;  
4  }
```

```
System.out.println(...)
```

```
System.out.println(...)
```



1. Methoden

Beispiele

- `System.out.println(...)`
- `Math.random()`

Wie funktioniert so eine Methode?

Mathematische Funktion

Mathematische Funktion

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

Mathematische Funktion

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

- $f(n) = \prod_{k=1}^n k$

Mathematische Funktion

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

- $f(n) = \prod_{k=1}^n k$

- Name: f

Mathematische Funktion

- $4! = 1 \cdot 2 \cdot 3 \cdot 4$
- $f(n) = \prod_{k=1}^n k$
- Name: f
- Eingabe: $n \in \mathbb{N}$

Mathematische Funktion

- $4! = 1 \cdot 2 \cdot 3 \cdot 4$
- $f(n) = \prod_{k=1}^n k$
- Name: f
- Eingabe: $n \in \mathbb{N}$
- Ausgabe: $f \rightarrow \mathbb{N}$

Mathematische Funktion

- $4! = 1 \cdot 2 \cdot 3 \cdot 4$
- $f(n) = \prod_{k=1}^n k$
- Name: f
- Eingabe: $n \in \mathbb{N}$
- Ausgabe: $f \rightarrow \mathbb{N}$
- Definition

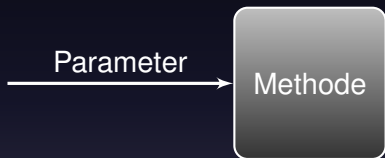
Black-Box

Black-Box



Methode

Black-Box



Black-Box



Beispiel: factorial

- Methodenname: `factorial`
- Parameter: `int n`
- Rückgabetyp: `int`

Wie rufe ich factorial auf?

Möglichkeiten des Methodenaufrufs

Möglichkeiten des Methodenaufrufs

- Einfach so:

Möglichkeiten des Methodenaufrufs

- Einfach so:

```
1 factorial(4);
```

Möglichkeiten des Methodenaufrufs

- Einfach so:

```
1 factorial(4);
```

- Speichern des Rückgabewerts in einer Variablen:

Möglichkeiten des Methodenaufrufs

- Einfach so:

```
1 factorial(4);
```

- Speichern des Rückgabewerts in einer Variablen:

```
1 int facFour;  
2 facFour = factorial(4);
```


Möglichkeiten des Methodenaufrufs

- Einfach so:

```
1 factorial(4);
```

- Speichern des Rückgabewerts in einer Variablen:

```
1 int facFour;  
2 facFour = factorial(4);
```

- Auswertung des Rückgabewerts in einem Ausdruck:

```
1 if ( factorial(4) == 24 ) {  
2     ...  
3 }
```

Syntax für den Aufruf

```
bezeichner(parameter, ...)
```

Wie schreibe ich eine neue Methode?

Syntax: Umgebung

Methoden gehören zu einer Klasse (`class`):

Syntax: Umgebung

Methoden gehören zu einer Klasse (class):

MathFunctions.java

```
1 public class MathFunctions {  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13 }
```

Syntax: Umgebung

Methoden gehören zu einer Klasse (class):

MathFunctions.java

```
1 public class MathFunctions {  
2     public static int factorial(int n) {  
3         ...  
4     }  
5  
6  
7  
8  
9  
10  
11  
12  
13 }
```

Syntax: Umgebung

Methoden gehören zu einer Klasse (class):

MathFunctions.java

```
1 public class MathFunctions {  
2     public static int factorial(int n) {  
3         ...  
4     }  
5  
6     public static int power(int base, int exp) {  
7         ...  
8     }  
9  
10  
11  
12  
13 }
```

Syntax: Umgebung

Methoden gehören zu einer Klasse (class):

MathFunctions.java

```
1 public class MathFunctions {
2     public static int factorial(int n) {
3         ...
4     }
5
6     public static int power(int base, int exp) {
7         ...
8     }
9
10    public static void main(String args[]) {
11        ...
12    }
13 }
```


Syntax: Aufbau

Syntax: Aufbau



Syntax: Aufbau

- Methoden-Kopf
 - enthält den Namen der Methode
 - enthält die Parameter
 - enthält den Rückgabebetyp



Syntax: Aufbau

- Methoden-Kopf
 - enthält den Namen der Methode
 - enthält die Parameter
 - enthält den Rückgabebetyp
- Methoden-Rumpf
 - ein Block ({ ... })
 - enthält die Funktion
 - gibt den Rückgabewert zurück



factorial, der Kopf

```
public static int factorial (int n) {  
    }  
}
```

factorial, der Kopf

```
public static int factorial (int n) {  
}
```

Rückgabotyp:
int

factorial, der Kopf

```
public static int factorial (int n) {  
}
```

Rückgabebetyp:
int

Name der Methode:
factorial

factorial, der Kopf

```
public static int factorial (int n) {  
}
```

The diagram illustrates the components of the method signature `public static int factorial (int n) {`. Three yellow ovals highlight the return type `int`, the method name `factorial`, and the parameter list `(int n)`. Arrows point from descriptive labels below to these ovals: an arrow from `Rückgabebetyp: int` points to the `int` oval; an arrow from `Name der Methode: factorial` points to the `factorial` oval; and an arrow from `Parameter: int n` points to the `(int n)` oval.

Rückgabebetyp:
int

Name der Methode:
factorial

Parameter:
int n

Syntax: Kopf

```
public static Typ methodenName (Typ name, ...) {  
    }  
}
```

Syntax: Kopf

```
public static Typ methodName (Typ name, ...) {  
}
```

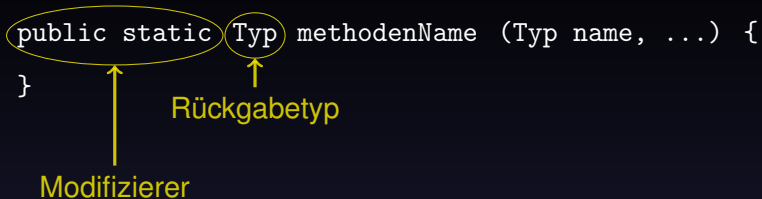
Modifizierer

Syntax: Kopf

`public static Typ methodenName (Typ name, ...) {`
`}`

↑
Modifizierer

↑
Rückgabebetyp

The diagram shows a Java method signature: `public static Typ methodenName (Typ name, ...) {` followed by a closing brace `}`. The words `public static` are enclosed in a yellow oval, with a yellow arrow pointing from the label `Modifizierer` below to this oval. The word `Typ` is also enclosed in a yellow oval, with a yellow arrow pointing from the label `Rückgabebetyp` below to this oval.

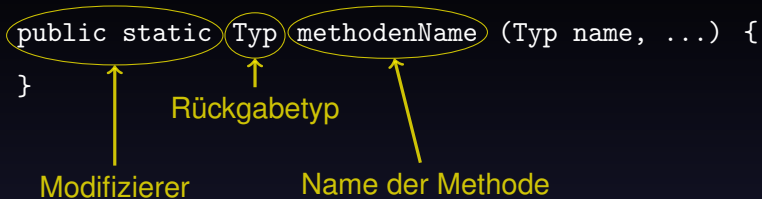
Syntax: Kopf

`public static` `Typ` `methodenName` (`Typ name, ...`) {
}

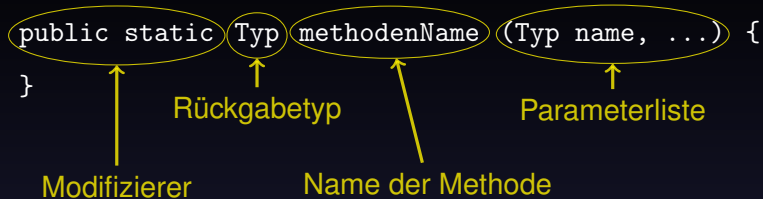
↑
Modifizierer

↑
Rückgabebetyp

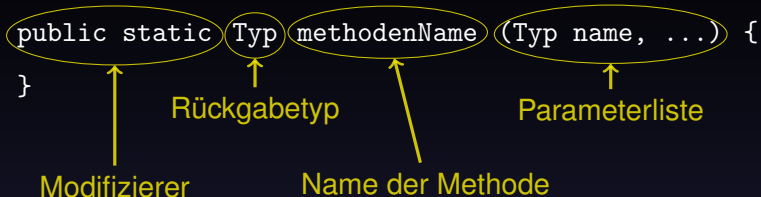
↑
Name der Methode

The diagram shows a Java method signature: `public static Typ methodenName (Typ name, ...) {`. The opening curly brace `{` is on the line below. Three yellow ovals highlight `public static`, `Typ`, and `methodenName`. Yellow arrows point from labels below to these ovals: 'Modifizierer' points to `public static`, 'Rückgabebetyp' points to `Typ`, and 'Name der Methode' points to `methodenName`.

Syntax: Kopf

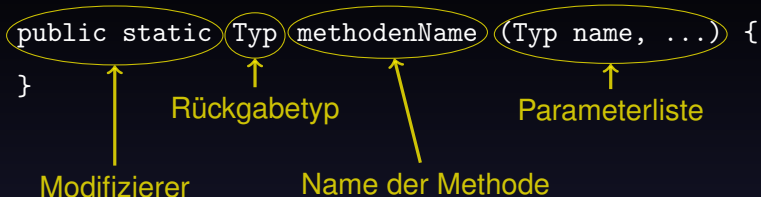


Syntax: Kopf



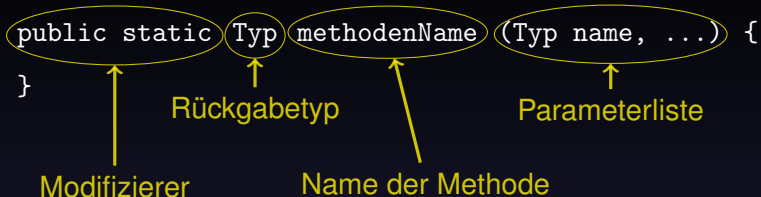
- mögliche Rückgabebetypen:
 - einfache Datentypen (z.B. `int`, `double`, ...)
 - komplexe Datentypen (z.B. `String`, `int []` (Arrays), ...)
 - `void` – keine Rückgabe

Syntax: Kopf



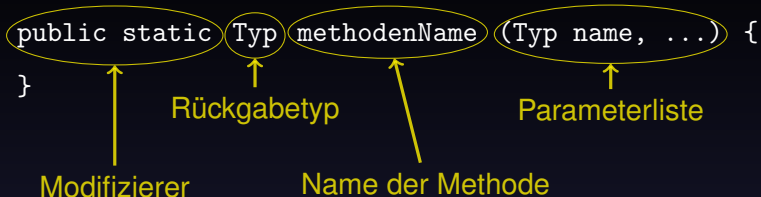
- mögliche Rückgabebetypen:
 - einfache Datentypen (z.B. `int`, `double`, ...)
 - komplexe Datentypen (z.B. `String`, `int []` (Arrays), ...)
 - `void` – keine Rückgabe
- Parameterliste kann 0 – ∞ Parameter enthalten

Syntax: Kopf



- mögliche Rückgabebetypen:
 - einfache Datentypen (z.B. `int`, `double`, ...)
 - komplexe Datentypen (z.B. `String`, `int []` (Arrays), ...)
 - `void` – keine Rückgabe
- Parameterliste kann $0 - \infty$ Parameter enthalten
- mögliche Parametertypen:

Syntax: Kopf



- mögliche Rückgabetypen:
 - einfache Datentypen (z.B. `int`, `double`, ...)
 - komplexe Datentypen (z.B. `String`, `int []` (Arrays), ...)
 - `void` – keine Rückgabe
- Parameterliste kann 0 – ∞ Parameter enthalten
- mögliche Parametertypen:
 - einfache Datentypen
 - komplexe Datentypen

Syntax: Kopf – Beispiele

Syntax: Kopf – Beispiele

```
1 public static void doSomething()
```

- keine Parameter
- keine Rückgabe (void)

Syntax: Kopf – Beispiele

```
1 public static void    doSomething()  
2 public static void    doSomething(int n)
```

- ein Parameter: `int n`
- keine Rückgabe (`void`)

Syntax: Kopf – Beispiele

```
1 public static void    doSomething()  
2 public static void    doSomething(int n)  
3 public static void    doSomething(int n, String s)
```

- zwei Parameter:

1 int n

2 String s

- keine Rückgabe (void)

Syntax: Kopf – Beispiele

```
1 public static void    doSomething()  
2 public static void    doSomething(int n)  
3 public static void    doSomething(int n, String s)  
4 public static int      doSomething()
```

- keine Parameter
- Rückgabe: `int`

Syntax: Kopf – Beispiele

```
1 public static void    doSomething()  
2 public static void    doSomething(int n)  
3 public static void    doSomething(int n, String s)  
4 public static int     doSomething()  
5 public static String  doSomething()
```

- keine Parameter
- Rückgabe: String

Syntax: Kopf – Beispiele

```
1 public static void    doSomething()  
2 public static void    doSomething(int n)  
3 public static void    doSomething(int n, String s)  
4 public static int     doSomething()  
5 public static String  doSomething()  
6 public static int[]   doSomething()
```

- keine Parameter
- Rückgabe: `int []` (Array von `int`)

Syntax: Kopf – Beispiele

```
1 public static void    doSomething()  
2 public static void    doSomething(int n)  
3 public static void    doSomething(int n, String s)  
4 public static int     doSomething()  
5 public static String  doSomething()  
6 public static int[]   doSomething()
```

Syntax: Rumpf

Syntax: Rumpf

```
1 public static int factorial(int n) {  
2     int result = 1;  
3     ... //result (Fakultaet von n) wird berechnet  
4     return result;  
5 }
```

Syntax: Rumpf

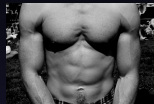
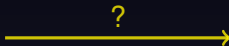
```
1 public static int factorial(int n) {  
2     int result = 1;  
3     ... //result (Fakultaet von n) wird berechnet  
4     return result;  
5 }
```

- `return` «Rückgabewert»;
 - bricht Ausführung ab und gibt «Rückgabewert» zurück
 - bei Rückgabety `void`: `return`;



?





Wie kommen die Parameter
vom Kopf in den Rumpf?

einfache Antwort:

einfache Antwort:

Sie werden hinein kopiert.

einfache Antwort:

Sie werden hinein kopiert.



Parameterübergabe

Parameterübergabe

```
1 public static int factorial(int n) {  
2  
3  
4  
5  
6  
7  
8 }
```

Parameterübergabe

```
1 public static int factorial(int n) {  
2     int result = 1;  
3     while(n != 0) {  
4  
5  
6     }  
7     return result;  
8 }
```

Parameterübergabe

```
1 public static int factorial(int n) {  
2     int result = 1;  
3     while(n != 0) {  
4         result = result * n;  
5         n = n - 1;  
6     }  
7     return result;  
8 }
```

Call by Value

Call by Value

```
main(...)
```


Call by Value

```
main(...)
```



Call by Value

main(...)



methode(...)

Call by Value

`main(...)`



copy



`methode(...)`

Call by Value

main(...)



methode(...)



Call by Value

main(...)



methode(...)



Call by Value: Beispiel

Call by Value: Beispiel

```
1 public class Modify {  
2     public static void main(String args[]) {  
3         int value = 42;  
4  
5         modify(value);  
6  
7     }  
8     public static void modify(int value) {  
9         value = 23;  
10  
11     }  
12 }
```

Call by Value: Beispiel

```
1 public class Modify {  
2     public static void main(String args[]) {  
3         int value = 42;  
4         System.out.println("before: " + value);  
5         modify(value);  
6         System.out.println("after: " + value);  
7     }  
8     public static void modify(int value) {  
9         value = 23;  
10        System.out.println("in modify: " + value);  
11    }  
12 }
```


Call by Value: Beispiel

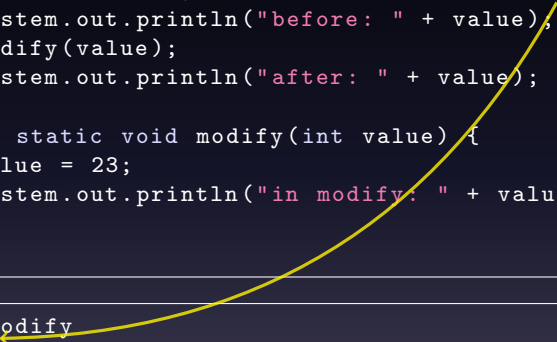
```
1 public class Modify {  
2     public static void main(String args[]) {  
3         int value = 42;  
4         System.out.println("before: " + value);  
5         modify(value);  
6         System.out.println("after: " + value);  
7     }  
8     public static void modify(int value) {  
9         value = 23;  
10        System.out.println("in modify: " + value);  
11    }  
12 }
```

```
1 ~ $ java Modify  
2 before: 42  
3 in modify: 23  
4 after: 42
```

Call by Value: Beispiel

```
1 public class Modify {  
2     public static void main(String args[]) {  
3         int value = 42;  
4         System.out.println("before: " + value);  
5         modify(value);  
6         System.out.println("after: " + value);  
7     }  
8     public static void modify(int value) {  
9         value = 23;  
10        System.out.println("in modify: " + value);  
11    }  
12 }
```

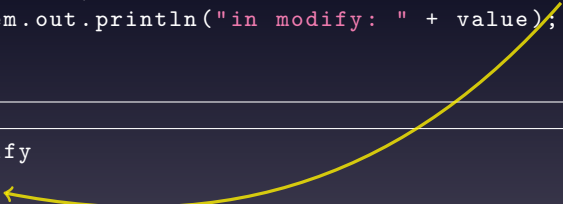
```
1 ~ $ java Modify  
2 before: 42  
3 in modify: 23  
4 after: 42
```



Call by Value: Beispiel

```
1 public class Modify {  
2     public static void main(String args[]) {  
3         int value = 42;  
4         System.out.println("before: " + value);  
5         modify(value);  
6         System.out.println("after: " + value);  
7     }  
8     public static void modify(int value) {  
9         value = 23;  
10        System.out.println("in modify: " + value);  
11    }  
12 }
```

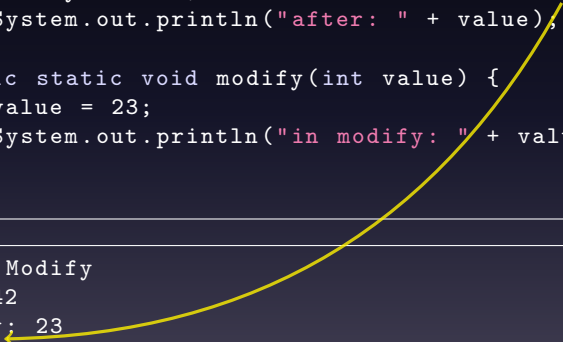
```
1 ~ $ java Modify  
2 before: 42  
3 in modify: 23  
4 after: 42
```



Call by Value: Beispiel

```
1 public class Modify {  
2     public static void main(String args[]) {  
3         int value = 42;  
4         System.out.println("before: " + value);  
5         modify(value);  
6         System.out.println("after: " + value);  
7     }  
8     public static void modify(int value) {  
9         value = 23;  
10        System.out.println("in modify: " + value);  
11    }  
12 }
```

```
1 ~ $ java Modify  
2 before: 42  
3 in modify: 23  
4 after: 42
```



Wäre da nicht ein Problem. . .

Wäre da nicht ein Problem. . .

Bei großen Datenmengen in den Parametern
muss alles komplett **kopiert** werden!



Call by Reference

Call by Reference

main(...)



methode(...)

Call by Reference

main(...)



methode(...)

Call by Reference

main(...)



copy

methode(...)



Call by Reference

`main(...)`



`methode(...)`



Call by Reference

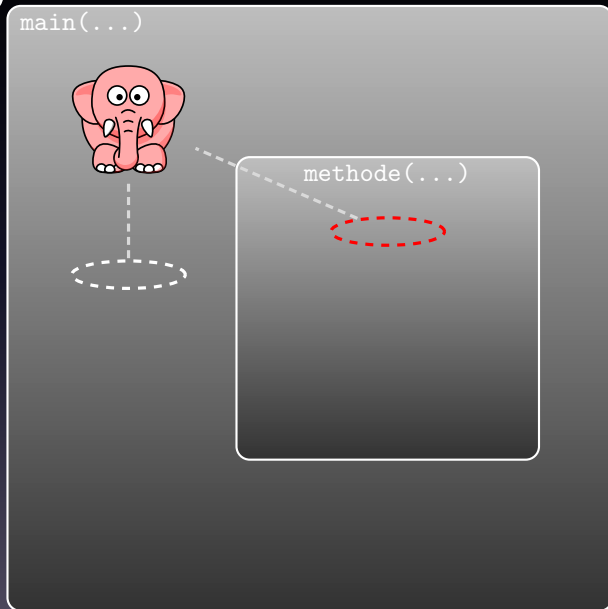
main(...)



methode(...)



Call by Reference



Call by Reference: Beispiel

Call by Reference: Beispiel

```
1 public class HugeCopy {
2     public static void main(String args[]) {
3         int [] arr = new int[10000];
4
5         setOne(arr);
6
7     }
8     public static void setOne(int q[]) {
9         for(int i=0; i<q.length; i++) {
10             q[i] = 1;
11         }
12
13     }
14 }
```


Call by Reference: Beispiel

```
1 public class HugeCopy {
2     public static void main(String args[]) {
3         int [] arr = new int[10000];
4         System.out.println("before: " + arr[9999]);
5         setOne(arr);
6         System.out.println("after: " + arr[9999]);
7     }
8     public static void setOne(int q[]) {
9         for(int i=0; i<q.length; i++) {
10             q[i] = 1;
11         }
12         System.out.println("in setOne: " + arr[9999]);
13     }
14 }
```

Call by Reference: Beispiel

```
1 public class HugeCopy {
2     public static void main(String args[]) {
3         int [] arr = new int[10000];
4         System.out.println("before: " + arr[9999]);
5         setOne(arr);
6         System.out.println("after: " + arr[9999]);
7     }
8     public static void setOne(int q[]) {
9         for(int i=0; i<q.length; i++) {
10             q[i] = 1;
11         }
12         System.out.println("in setOne: " + arr[9999]);
13     }
14 }
```

```
1 ~ $ java HugeCopy
2 before: 0
3 in setOne: 1
4 after: 1
```

Call by Reference: Beispiel


```
1 public class HugeCopy {
2     public static void main(String args[]) {
3         int [] arr = new int[10000];
4         System.out.println("before: " + arr[9999]);
5         setOne(arr);
6         System.out.println("after: " + arr[9999]);
7     }
8     public static void setOne(int q[]) {
9         for(int i=0; i<q.length; i++) {
10             q[i] = 1;
11         }
12         System.out.println("in setOne: " + arr[9999]);
13     }
14 }
```

```
1 ~ $ java HugeCopy
2 before: 0
3 in setOne: 1
4 after: 1
```

Call by Reference: Beispiel

```
1 public class HugeCopy {
2     public static void main(String args[]) {
3         int [] arr = new int[10000];
4         System.out.println("before: " + arr[9999]);
5         setOne(arr);
6         System.out.println("after: " + arr[9999]);
7     }
8     public static void setOne(int q[]) {
9         for(int i=0; i<q.length; i++) {
10             q[i] = 1;
11         }
12         System.out.println("in setOne: " + arr[9999]);
13     }
14 }
```

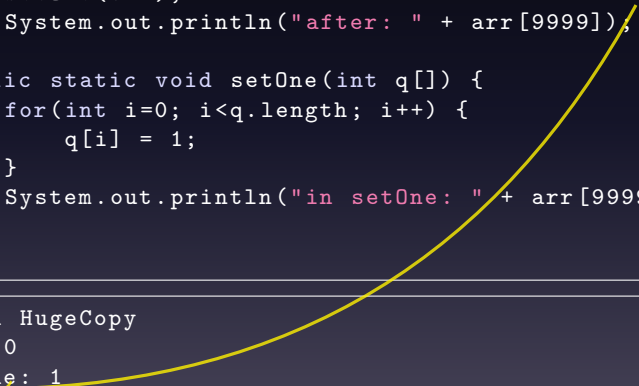
```
1 ~ $ java HugeCopy
2 before: 0
3 in setOne: 1
4 after: 1
```



Call by Reference: Beispiel

```
1 public class HugeCopy {
2     public static void main(String args[]) {
3         int [] arr = new int[10000];
4         System.out.println("before: " + arr[9999]);
5         setOne(arr);
6         System.out.println("after: " + arr[9999]);
7     }
8     public static void setOne(int q[]) {
9         for(int i=0; i<q.length; i++) {
10             q[i] = 1;
11         }
12         System.out.println("in setOne: " + arr[9999]);
13     }
14 }
```

```
1 ~ $ java HugeCopy
2 before: 0
3 in setOne: 1
4 after: 1
```



Call by Reference vs. Call by Value

- richtet sich nach Datentyp (automatisch)
- Call by Value
 - Kopieren der Parameter
 - für einfache Datentypen (`int`, `double`, `float`, `char`, ...)
- Call by Reference
 - Referenzieren der Parameter
 - für komplexe Datentypen
 - z.B. Arrays

2. Testen

Was heißt Testen?

Was kann man Testen?

Was kann man Testen?

Methoden

Wie Testen?

Der Idealfall:

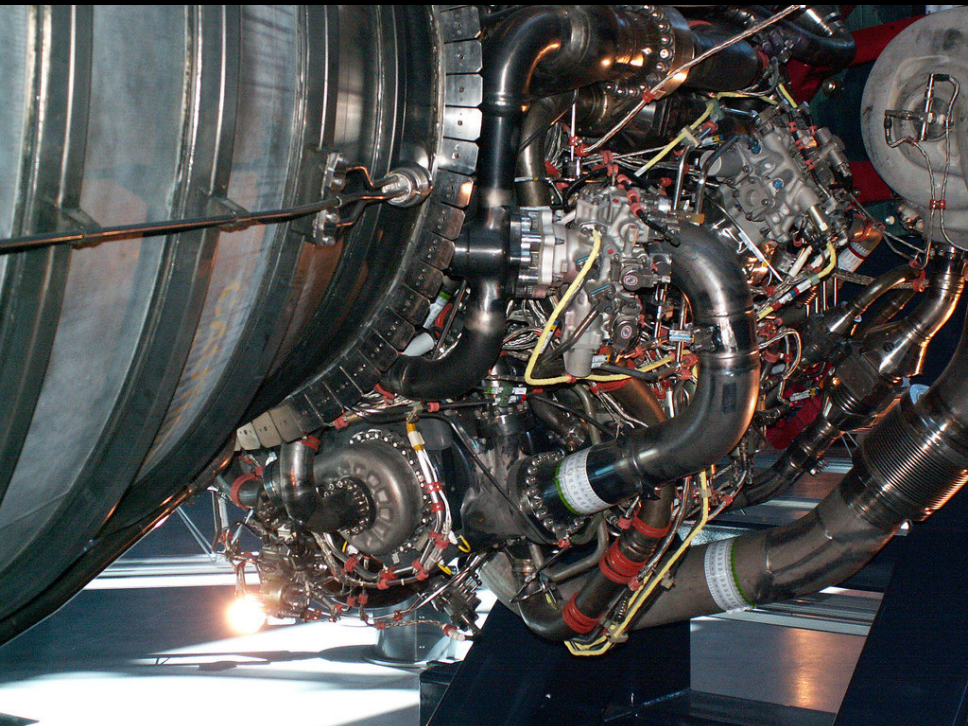
- 1 Vorstellung davon was eine Method tun soll
- 2 Methoden-Kopf erstellen
- 3 Testfälle schreiben
- 4 Methode implementieren
- 5 Testfälle aufrufen

Warum Testen?



TLE LAUNCH
N FACILITY







Warum Testen?

Vorher Testen ist schneller
als hinterher Fehler zu suchen

denn:

Fehler sind meist schwer zu finden

Wie sollte ein Test aussehen?

Wie sollte ein Test aussehen?

Factorial.java

```
1 public static int factorial(int n) {return 0;}
```

2

3

4

5

6

7

8

9

10

11

Wie sollte ein Test aussehen?

Factorial.java

```
1 public static int factorial(int n) {return 0;}
2
3 public static void testFactorial() {
4
5
6
7 }
8
9 public static void main(String args[]) {
10     testFactorial();
11 }
```

Wie sollte ein Test aussehen?

Factorial.java

```
1 public static int factorial(int n) {return 0;}
2
3 public static void testFactorial() {
4
5
6
7 }
8
9 public static void main(String args[]) {
10     testFactorial();
11 }
```

```
1 ~ $ java Factorial
2 factorial(4) expected: 24 result: 0
3 factorial(1) expected: 1 result: 0
4 factorial(0) expected: 1 result: 0
```

Wie sollte ein Test aussehen?

Factorial.java

```
1 public static int factorial(int n) {return 0;}
2
3 public static void testFactorial() {
4     printTest("factorial", 4, factorial(4), 24);
5     printTest("factorial", 1, factorial(1), 1);
6     printTest("factorial", 0, factorial(0), 1);
7 }
8
9 public static void main(String args[]) {
10     testFactorial();
11 }
```

```
1 ~ $ java Factorial
2 factorial(4) expected: 24 result: 0
3 factorial(1) expected: 1 result: 0
4 factorial(0) expected: 1 result: 0
```

printTest

```
1 public static void printTest(  
2     String methodName,  
3     int param,  
4     int result,  
5     int expected) {  
6  
7     System.out.println(  
8         methodName +  
9         "(" + param + ")" +  
10        " expected: " + expected +  
11        " result: " + result  
12    );  
13 }
```

```
1 ~ $ java Factorial  
2 factorial(4) expected: 24 result: 0  
3 factorial(1) expected: 1 result: 0  
4 factorial(0) expected: 1 result: 0
```

Factorial implementiert, 1. Versuch

```
1 public static int factorial(int n) {  
2     int fac = 1;  
3     while(n != 0) {  
4         fac = fac * n;  
5         n = n - 1;  
6     }  
7     return fac;  
8 }
```


Factorial, 1. Versuch, Test

```
1 ~ $ java Factorial
2 factorial(4) expected: 24 result: 24
3 factorial(1) expected: 1 result: 1
4 factorial(0) expected: 1 result: 1
```

Factorial: mehr Tests

```
1 public static void testFactorial() {  
2     printTest("factorial", 4, factorial(4), 24);  
3     printTest("factorial", 1, factorial(1), 1);  
4     printTest("factorial", 0, factorial(0), 1);  
5     printTest("factorial", -1, factorial(-1), 0);  
6 }
```

Was passiert?

Factorial, Test

```
1 ~ $ java Factorial
2 factorial(4) expected: 24 result: 24
3 factorial(1) expected: 1 result: 1
4 factorial(0) expected: 1 result: 1
5 -
```

Factorial, Test

```
1 ~ $ java Factorial
2 factorial(4) expected: 24 result: 24
3 factorial(1) expected: 1 result: 1
4 factorial(0) expected: 1 result: 1
5 -
```



...Stunden später ...

$$-1!$$

Factorial implementiert, 2. Versuch

```
1 public static int factorial(int n) {  
2     if(n<0){return 0;}  
3     int fac = 1;  
4     while(n != 0) {  
5         fac = fac * n;  
6         n = n - 1;  
7     }  
8     return fac;  
9 }
```

Factorial implementiert, 2. Versuch

```
1 public static int factorial(int n) {  
2     if(n<0){return 0;}  
3     int fac = 1;  
4     while(n != 0) {  
5         fac = fac * n;  
6         n = n - 1;  
7     }  
8     return fac;  
9 }
```

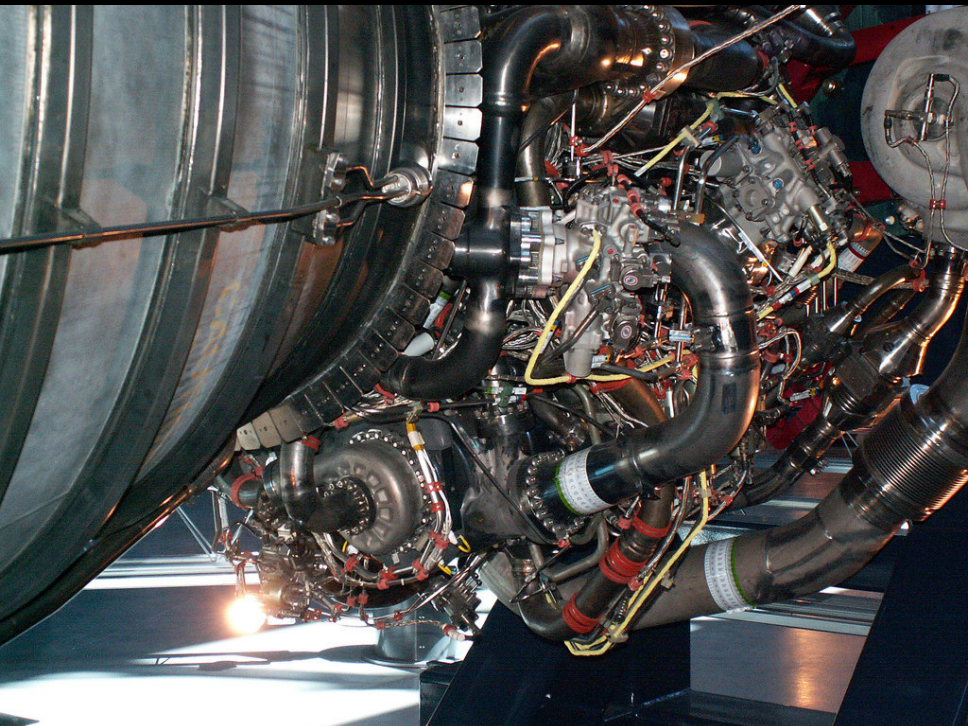
```
1 ~ $ java Factorial  
2 factorial(4) expected: 24 result: 24  
3 factorial(1) expected: 1 result: 1  
4 factorial(0) expected: 1 result: 1  
5 factorial(-1) expected: 0 result: 0
```


Grundsätze zum Testen

- Erst den Test, dann die Implementierung
- typische Fälle testen
- Randbereiche testen
- Sonderfälle testen
- Viel hilft Viel!

5. Debugging





(Debugging == Wie finde ich die lose
Schraube?)

Systematik

- Fehlerstelle eingrenzen
- Programmablauf überprüfen

Systematik

- Fehlerstelle eingrenzen
- Programmablauf überprüfen
- durch: Kontrollausgaben

Beispiel - Modulo

```
1 public static int modulo(int zahl, int divisor) {  
2     int modulo = zahl;  
3  
4     while(modulo > divisor) {  
5  
6  
7         modulo = modulo - divisor;  
8     }  
9  
10    return modulo;  
11 }
```

Beispiel - Modulo

Code wurde nicht getestet

Beispiel - Modulo

Code wurde nicht getestet

Falsches Ergebnis

ohne Kontrollausgaben

```
1 public static int modulo(int zahl, int divisor) {  
2     int modulo = zahl;  
3  
4     while(modulo > divisor) {  
5  
6  
7         modulo = modulo - divisor;  
8     }  
9  
10    return modulo;  
11 }
```

mit Kontrollausgaben

```
1 public static int modulo(int zahl, int divisor) {  
2     int modulo = zahl;  
3     System.out.println(zahl + " % " + divisor);  
4     while(modulo > divisor) {  
5         System.out.print("modulo - divisor: " + modulo +  
6             " - " + divisor + " = " + (modulo - divisor));  
7         modulo = modulo - divisor;  
8     }  
9     System.out.println(zahl+" % "+divisor+" = " + modulo);  
10    return modulo;  
11 }
```

Ausgaben

Ausgaben

```
1 6 % 2
2 modulo - divisor: 6 - 2 = 4
3 modulo - divisor: 4 - 2 = 2
4 6 % 2 = 2
```

Ausgaben

```
1 6 % 2
2 modulo - divisor: 6 - 2 = 4
3 modulo - divisor: 4 - 2 = 2
4 6 % 2 = 2
```

- `modulo(6, 2)` sollte 0 sein
- Wo ist der Fehler?
 - Es wurde 1x zu wenig modulo abgezogen
 - Vergleich ist falsch
 - 4. Zeile: > Sollte >=

Debugging - Lösung

```
1 public static int modulo(int zahl, int divisor) {  
2     int modulo = zahl;  
3     //System.out.println(zahl + "%" + divisor);  
4     while(modulo >= divisor) {  
5         //System.out.print("modulo - divisor: " + modulo +  
6         // "- " divisor + " = " + (modulo -divisor));  
7         modulo = modulo - divisor;  
8     }  
9     //System.out.println(zahl+"%" +divisor+"=" + modulo);  
10    return modulo;  
11 }
```


3. Java-API

A 4x4 grid of 16 photographs showing the contents of a cluttered bookshelf. The shelves are filled with books, CDs, DVDs, a box of tissues, a coffee maker, a small framed photo, a telephone, a red box, a bag, a box of photos, and various other miscellaneous items.

Java-API

- Standard-Funktionen:
 - Konsolenausgaben
 - Mathematische Berechnungen
 - Datenstrukturen (Listen, Bäume)
 - ...



Java-API

Wie finde ich diese Standard-Funktionen?

Java-API



java api 6

Google-Suche

Auf gut Glück!

Java-API



java api 6

Google-Suche

Auf gut Glück!

[Java Platform SE 6](#) [[Diese Seite übersetzen](#)]

Frame Alert. This document is designed to be viewed using the frames feature. If you see this message, you are using a non-frame-capable web client. ...

java.sun.com/javase/6/docs/api/ - 2k - [Im Cache](#) - [Ähnliche Seiten](#)

Java-API - Übersicht

Java™ Platform
Standard Ed. 6

[All Classes](#)

Packages
[java.applet](#)
[java.awt](#)
[java.awt.color](#)
[java.awt.datatransfer](#)
[java.awt.dnd](#)

All Classes
[AbstractAction](#)
[AbstractAnnotationValueVis](#)
[AbstractBorder](#)
[AbstractButton](#)
[AbstractCellEditor](#)
[AbstractCollection](#)
[AbstractColorChooserPanel](#)
[AbstractDocument](#)
[AbstractDocument.Attribute](#)
[AbstractDocument.Content](#)
[AbstractDocument.Element](#)
[AbstractElementVisitor6](#)
[AbstractExecutorService](#)
[AbstractInterruptibleChanne](#)
[AbstractLayoutCache](#)
[AbstractLayoutCache.Node](#)
[AbstractList](#)
[AbstractListModel](#)
[AbstractMap](#)
[AbstractMap.SimpleEntry](#)
[AbstractMap.SimpleImmuta](#)
[AbstractMarshallerImpl](#)
[AbstractMethodError](#)
[AbstractOwnableSynchroniz](#)
[AbstractPreferences](#)

Overview Package Class Use Tree Deprecated Index Help

PREV NEXT FRAMES NO FRAMES

Java™ Platform, Standard Edition 6
API Specification

This document is the API specification for version 6 of the Java™ Platform, Standard Edition.

See:
[Description](#)

Packages	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.

Exkurs: Package

Exkurs: Package



Exkurs: Package



Exkurs: Package



- Ähnlich einer Verzeichnisstruktur

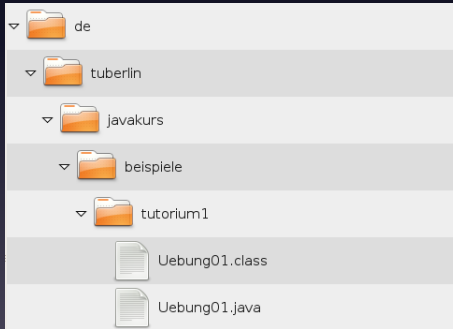
Exkurs: Package



- Ähnlich einer Verzeichnisstruktur
- Strukturierung nach unterschiedlichen Gesichtspunkten,

Exkurs: Package

```
1 package de.tuberlin.javakurs.beispiele.tutorium1;  
2  
3 public class Uebung01 {  
4     ...  
5 }
```



Java-API - Übersicht

Java™ Platform
Standard Ed. 6

[All Classes](#)

Packages
[java.applet](#)
[java.awt](#)
[java.awt.color](#)
[java.awt.datatransfer](#)
[java.awt.dnd](#)

All Classes
[AbstractAction](#)
[AbstractAnnotationValueVis](#)
[AbstractBorder](#)
[AbstractButton](#)
[AbstractCellEditor](#)
[AbstractCollection](#)
[AbstractColorChooserPanel](#)
[AbstractDocument](#)
[AbstractDocument.Attribute](#)
[AbstractDocument.Content](#)
[AbstractDocument.Element](#)
[AbstractElementVisitor6](#)
[AbstractExecutorService](#)
[AbstractInterruptibleChanne](#)
[AbstractLayoutCache](#)
[AbstractLayoutCache.Node](#)
[AbstractList](#)
[AbstractListModel](#)
[AbstractMap](#)
[AbstractMap.SimpleEntry](#)
[AbstractMap.SimpleImmuta](#)
[AbstractMarshallerImpl](#)
[AbstractMethodError](#)
[AbstractOwnableSynchroniz](#)
[AbstractPreferences](#)

Overview Package Class Use Tree Deprecated Index Help

PREV NEXT FRAMES NO FRAMES

Java™ Platform, Standard Edition 6
API Specification

This document is the API specification for version 6 of the Java™ Platform, Standard Edition.

See:
[Description](#)

Packages	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.

Java-API - Math.random()

**Java™ Platform
Standard Ed. 6**

Overview Package Class Use Tree Deprecated Index Help

PREV NEXT FRAMES NO FRAMES

Java™ Platform, Standard Edition 6 API Specification

This document is the API specification for version 6 of the Java™ Platform, Standard Edition.

See:

[Description](#)

Packages	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
javax.swing	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
javax.swing.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
javax.swing.text	Provides classes and interfaces for editing text.

[MatchResult](#)

[Math](#)

[MathContext](#)

[Matcher](#)

[MD5](#)

[MBeanNotificationInfo](#)

Java-API - Math.random()

Java™ Platform
Standard Ed. 6

[All Classes](#)

Packages

- [java.applet](#)
- [java.awt](#)
- [java.awt.color](#)
- [java.awt.datatransfer](#)
- [java.awt.dnd](#)

Management/Permission
ManageReferralControl
ManagerFactoryParameters
Manifest
Manifest
Map
Map.Entry
MappedByteBuffer
MARSHAL
MarshalException
MarshalException
MarshaledObject
Marshaller
Marshaller.Listener
MaskFormatter
Matcher

[MatchResult](#)

[Math](#)

[MathContext](#)

[Math class in java.lang](#)

MathException
MathNotificationInfo

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)
[FRAMES](#) [NO FRAMES](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

java.lang

Class Math

[java.lang.Object](#)
└─ [java.lang.Math](#)

public final class **Math**
extends [Object](#)

The class **Math** contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

Unlike some of the numeric methods of class **StrictMath**, all implementations of the equivalent functions of class **Math** are not defined to return the bit-for-bit same results. This relaxation permits better-performing implementations where strict reproducibility is not required.

By default many of the **Math** methods simply call the equivalent method in **StrictMath** for their implementation. Code generators are encouraged to use platform-specific native libraries or microprocessor instructions, where available, to provide higher-performance implementations of **Math** methods. Such higher-performance implementations still must conform to the specification for **Math**.

The quality of implementation specifications concern two properties, accuracy of the returned result and monotonicity of the method. Accuracy of the floating-point **Math** methods is defined in terms of *ulps*, units in the last place. For a given floating-point format, an *ulp* of a real number value is the distance between the two floating-point values *g* that numerical value. When discussing the accuracy of a method as a whole or at a specific argument, the number of *ulps* cited is for the worst-case error at that argument. If a method always has an error less than 0.5 *ulps*, the method always returns the floating-point number nearest the exact result; such a method is *correctly rounded*. A correctly rounded method is generally the best a floating-point approximation method. However, it is impractical for many floating-point methods to be correctly rounded. For the **Math** class, a larger error bound of 1 or 2 *ulps* is allowed for certain methods. Informally, with a 1 *ulp* error bound, when the exact result is a representable number, the

Java™ Platform
Standard Ed. 6

Java-API - Math.random()

Java™ Platform
Standard Ed. 6

[All Classes](#)

Packages
[java.applet](#)
[java.awt](#)
[java.awt.color](#)
[java.awt.datatransfer](#)
[java.awt.dnd](#)

ManagementPermission
ManageReferralControl
ManagerFactoryParameters
Manifest
Manifest
Map
Map.Entry
MappedByteBuffer
MARSHAL
MarshalException
MarshalException
MarshalException
MarshalledObject
Marshaller
Marshaller.Listener
MaskFormatter
Matcher

[MatchResult](#)
[Math](#)
[MathContext](#)
[Math](#) class in java.lang

MBEANAttributeInfo
MBEANNotificationInfo

	direction of the second argument.
static float	nextAfter (float start, double direction) Returns the floating-point number adjacent to the first argument in the direction of the second argument.
static double	nextUp (double d) Returns the floating-point value adjacent to d in the direction of positive infinity.
static float	nextUp (float f) Returns the floating-point value adjacent to f in the direction of positive infinity.
static double	pow (double a, double b) Returns the value of the first argument raised to the power of the second argument.
static double	random () Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static double	rint (double a) Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
static long	round (double a) Returns the closest long to the argument.
static int	round (float a) Returns the closest int to the argument.
static double	scalb (double d, int scaleFactor) Return $d \times 2^{\text{scaleFactor}}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set.
static float	scalb (float f, int scaleFactor) Return $f \times 2^{\text{scaleFactor}}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the float value set.
	signum (double d) Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.
	signum (float f) Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.
static double	sin (double a) Returns the trigonometric sine of a in radians.

Java-API - Math.random()

static double [random\(\)](#)

Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

Java-API - Math.random()

<code>static double</code>	<code>random()</code> Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
----------------------------	---

- Bezeichnung

Java-API - Math.random()

<code>static double</code>	<code>random()</code> Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
----------------------------	---

- Bezeichnung
- Beschreibung

Java-API - Math.random()

<code>static double</code>	<code>random()</code> Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
----------------------------	---

- Bezeichnung
- Beschreibung
- Rückgabewert und Typ

Java-API - Math.pow()

```
static double pow(double a, double b)
```

Returns the value of the first argument raised to the power of the second argument.

Java-API - Math.pow()

```
static double pow(double a, double b)
```

Returns the value of the first argument raised to the power of the second argument.

- Parameter (Anzahl und Typen)
 - double a, double b

Fragen?

Viel Spaß bei den
Übungen!

Bildquellen

Dank an / Thanks to:

Name: Steve Berry URL: www.flickr.com/photos/unloveable/2387650243/

Name: Marc Worrel URL: www.flickr.com/photos/mworrell/266913194/

Name: jonrawlinson URL: www.flickr.com/photos/london/45795719/

Name: Meredith Farmer: URL: www.flickr.com/photos/meredithfarmer/448440646/

Name: Jennifer Woodard Maderazo URL: www.flickr.com/photos/jenniferwoodardmaderazo/492252645/

Name: Thierry URL: www.flickr.com/photos/http2007/2376457040/

Name: Tim Ellis URL: www.flickr.com/photos/tim_ellis/541467303/

Name: mocr URL: www.flickr.com/photos/alpha262/149949262/

Name: Chris Christner URL: www.flickr.com/photos/toptechwriter/44176435/

Name: Meg Wills URL: www.flickr.com/photos/whatmegsaid/3209662385/

Name: striatic URL: www.flickr.com/photos/striatic/729822/