Javakurs 2009 – LE3

Methoden, Testen, Debuggen

Kai Dietrich, Dennis Guse

25. März 2009

Agenda

1. Methoden
2. Testen
3. Java-API
4. Namensgebung
5. Debuggen

0. Wiederholung

Wiederholung

Wiederholung

- Variablen und Zuweisungen

Wiederholung

- Variablen und Zuweisungen

```
int foo = 42;
String text = "Hallo Welt!";

int bar;
bar = 23;
```

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen

```
if ( heuteIstRasenmaehertag == True ) {
        System.out.println("Geh Rasen maehen!");
else {
        System.out.println("Faulenzen!");
}
```

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen

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

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen

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

```java
System.out.println("Ich");
int count = 0;
while(count<10) {
        System.out.println("maehe");
        count++;
}
```

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen
- Arrays

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen
- Arrays

```
int [] grashalme = new int[10];
grashalme[0] = 0;
grashalme[1] = 0;
grashalme[2] = 0;
...
grashalme[9] = 0;
```

Wiederholung

- Variablen und Zuweisungen
- Verzweigungen
- Schleifen
- Arrays

```
int [] grashalme = new int[10];
grashalme[0] = 0;
grashalme[1] = 0;
grashalme[2] = 0;
...
grashalme[9] = 0;
```

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

```
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: $\mathbb{N}$

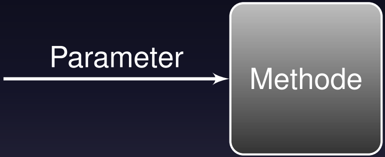
Mathematische Funktion

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

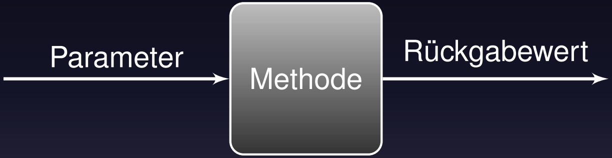
Black-Box

Black-Box

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:

```
factorial(4);
```

Möglichkeiten des Methodenaufrufs

- Einfach so:

```
factorial(4);
```

- Speichern des Rückgabewerts in einer Variablen:

Möglichkeiten des Methodenaufrufs

- Einfach so:

```
factorial(4);
```

- Speichern des Rückgabewerts in einer Variablen:

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

Möglichkeiten des Methodenaufrufs

- Einfach so:

```
factorial(4);
```

- Speichern des Rückgabewerts in einer Variablen:

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

- Auswertung des Rückgabewerts in einem Ausdruck:

```
if ( factorial(4) == 24 ) {
        ...
}
```

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

```java
public class MathFunctions {











}
```

Syntax: Umgebung

Methoden gehören zu einer Klasse (`class`):

MathFunctions.java

```java
public class MathFunctions {
        public static int factorial(int n) {
                ...
        }









}
```

Syntax: Umgebung

Methoden gehören zu einer Klasse (`class`):

MathFunctions.java

```
public class MathFunctions {
        public static int factorial(int n) {
                ...
        }

        public static int power(int base, int exp) {
                ...
        }




}
```

Syntax: Umgebung

Methoden gehören zu einer Klasse (`class`):

MathFunctions.java

```java
public class MathFunctions {
        public static int factorial(int n) {
                ...
        }

        public static int power(int base, int exp) {
                ...
        }

        public static void main(String args[]) {
                ...
        }
}
```

Syntax: Aufbau

Syntax: Aufbau

Syntax: Aufbau

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

Syntax: Aufbau

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

- 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) {
}
```

Rückgabetyp:
`int`

factorial, der Kopf

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

Rückgabetyp:
`int`

Name der Methode:
`factorial`

factorial, der Kopf

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

Rückgabetyp:
`int`

Parameter:
`int n`

Name der Methode:
`factorial`

Syntax: Kopf

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

Syntax: Kopf

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

}
```

Rückgabetyp

Syntax: Kopf

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

}
```

Rückgabetyp

Name der Methode

Syntax: Kopf

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

}
```

Rückgabetyp

Parameterliste

Name der Methode

Syntax: Kopf

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

}
```

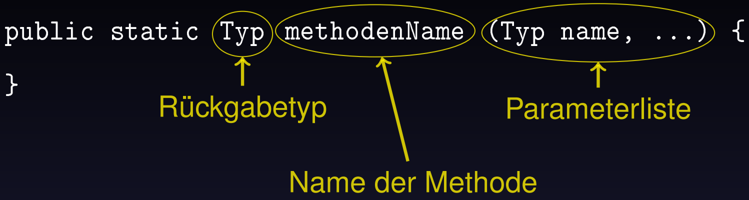
Rückgabetyp

Parameterliste

Name der Methode

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

Syntax: Kopf



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

}
```

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

Syntax: Kopf

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

}
```

Rückgabetyp

Parameterliste

Name der Methode

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

Syntax: Kopf – Beispiele

Syntax: Kopf – Beispiele

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

- keine Parameter
- keine Rückgabe (`void`)

Syntax: Kopf – Beispiele

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

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

Syntax: Kopf – Beispiele

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

- zwei Paramter:
  1. `int n`
  2. `String s`
- keine Rückgabe (`void`)

Syntax: Kopf – Beispiele

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

- keine Parameter
- Rückgabe: `int`

Syntax: Kopf – Beispiele

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

- keine Parameter
- Rückgabe: `String`

Syntax: Kopf – Beispiele

```
public static void   doSomething()
public static void   doSomething(int n)
public static void   doSomething(int n, String s)
public static int    doSomething()
public static String doSomething()
public static int[]  doSomething()
```

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

Syntax: Kopf – Beispiele

```
public static void   doSomething()
public static void   doSomething(int n)
public static void   doSomething(int n, String s)
public static int    doSomething()
public static String doSomething()
public static int[]  doSomething()
```

Syntax: Rumpf

Syntax: Rumpf

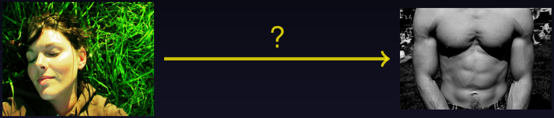
```
public static int factorial(int n) {
        int result = 1;
        ... //result (Fakultaet von n) wird berechnet
        return result;
}
```

Syntax: Rumpf

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

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

$$\xrightarrow{?}$$



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

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






}
```

Parameterübergabe

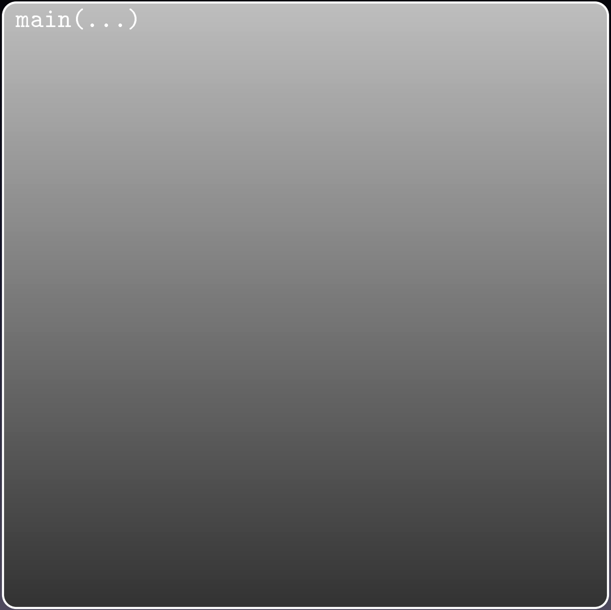
```java
public static int factorial(int n) {
    int fac = 1;
    while(n != 0) {


    }
    return fac;
}
```

Parameterübergabe

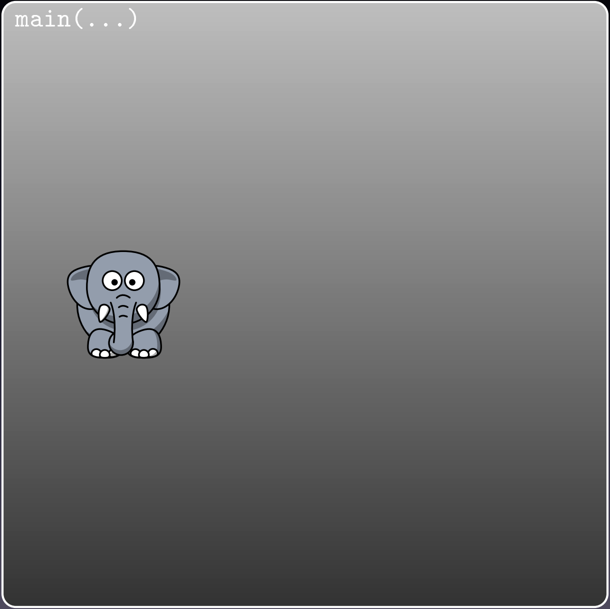
```java
public static int factorial(int n) {
    int fac = 1;
    while(n != 0) {
        fac = fac * n;
        n = n - 1;
    }
    return fac;
}
```

Call by Value

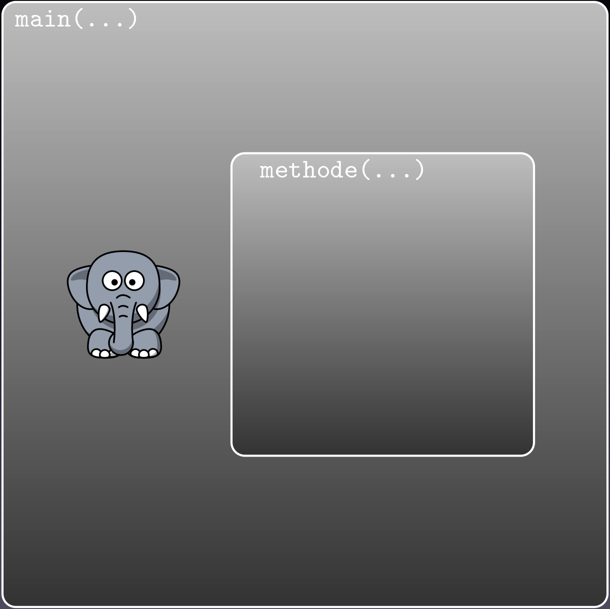
Call by Value



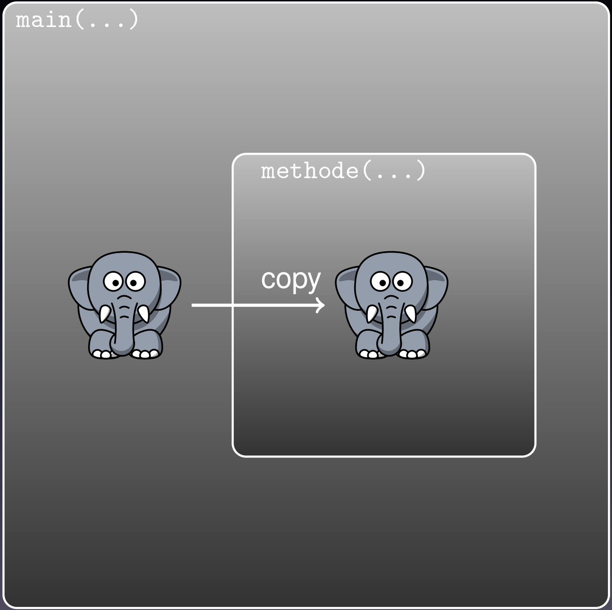
Call by Value



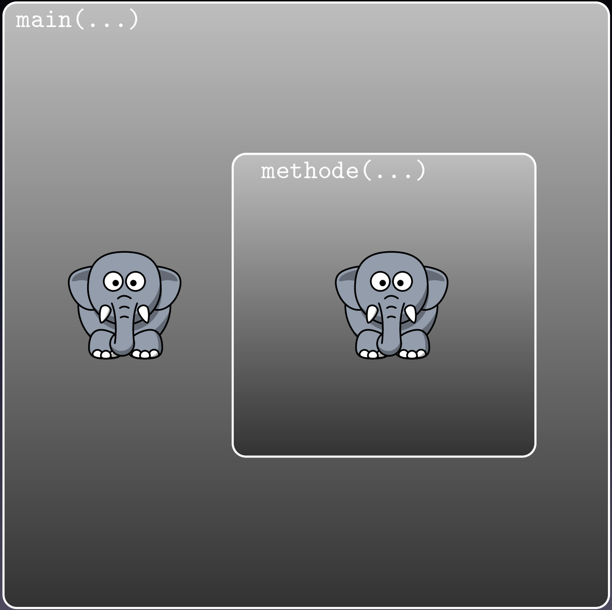
Call by Value



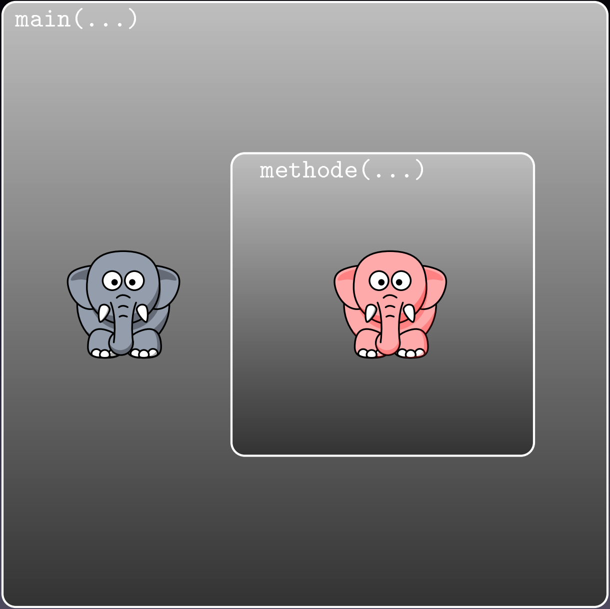
Call by Value



Call by Value



Call by Value



Call by Value: Beispiel

Call by Value: Beispiel

```java
public class Modify {
    public static void main(String args[]) {
        int value = 42;

        modify(value);

    }
    public static void modify(int value) {
        value = 23;

    }
}
```

Call by Value: Beispiel

```java
public class Modify {
    public static void main(String args[]) {
        int value = 42;
        System.out.println("before: " + value);
        modify(value);
        System.out.println("after: " + value);
    }
    public static void modify(int value) {
        value = 23;
        System.out.println("in modify: " + value);
    }
}
```

Call by Value: Beispiel

```java
public class Modify {
    public static void main(String args[]) {
        int value = 42;
        System.out.println("before: " + value);
        modify(value);
        System.out.println("after: " + value);
    }
    public static void modify(int value) {
        value = 23;
        System.out.println("in modify: " + value);
    }
}
```

```
~ $ java Modify
before: 42
in modify: 23
after: 42
```

Call by Value: Beispiel

```java
public class Modify {
    public static void main(String args[]) {
        int value = 42;
        System.out.println("before: " + value);
        modify(value);
        System.out.println("after: " + value);
    }
    public static void modify(int value) {
        value = 23;
        System.out.println("in modify: " + value);
    }
}
```

```
~ $ java Modify
before: 42
in modify: 23
after: 42
```

Call by Value: Beispiel

```java
public class Modify {
    public static void main(String args[]) {
        int value = 42;
        System.out.println("before: " + value);
        modify(value);
        System.out.println("after: " + value);
    }
    public static void modify(int value) {
        value = 23;
        System.out.println("in modify: " + value);
    }
}
```

```
~ $ java Modify
before: 42
in modify: 23
after: 42
```

Call by Value: Beispiel

```
public class Modify {
    public static void main(String args[]) {
        int value = 42;
        System.out.println("before: " + value);
        modify(value);
        System.out.println("after: " + value);
    }
    public static void modify(int value) {
        value = 23;
        System.out.println("in modify: " + value);
    }
}
```

```
~ $ java Modify
before: 42
in modify: 23
after: 42
```

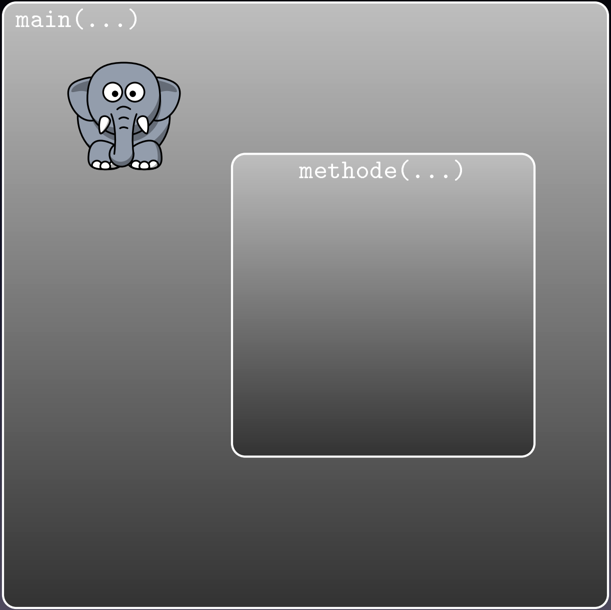
Wäre da nicht ein Problem. . .

Wäre da nicht ein Problem. . .

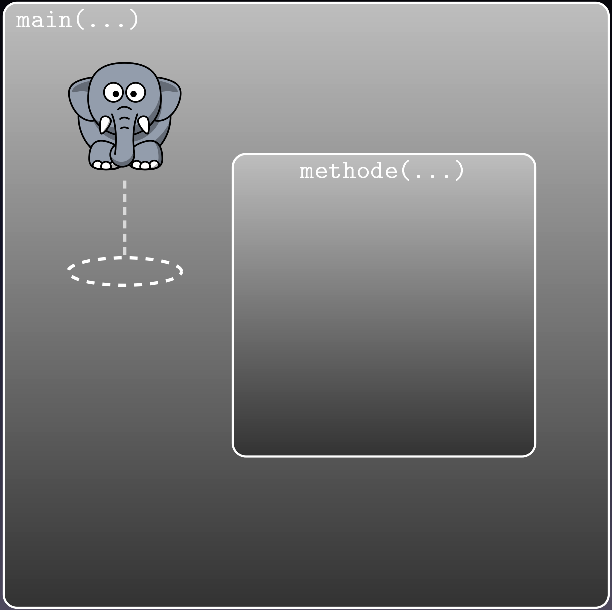
Bei großen Datenmengen in den Parametern muss alles komplett **kopiert** werden!

Call by Reference

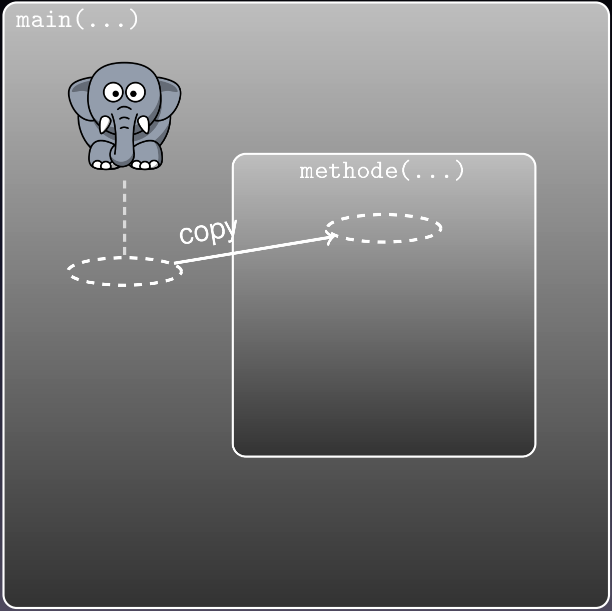
Call by Reference



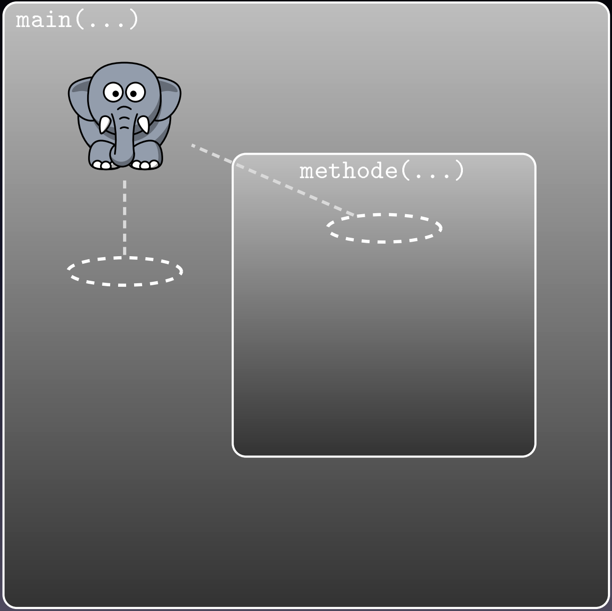
Call by Reference



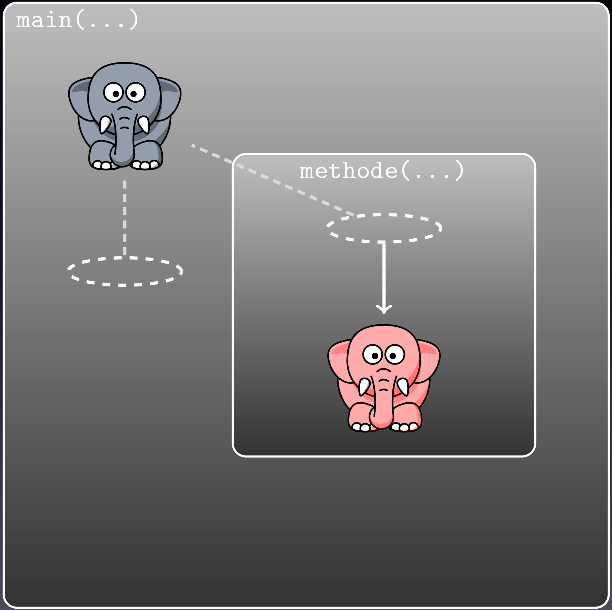
Call by Reference



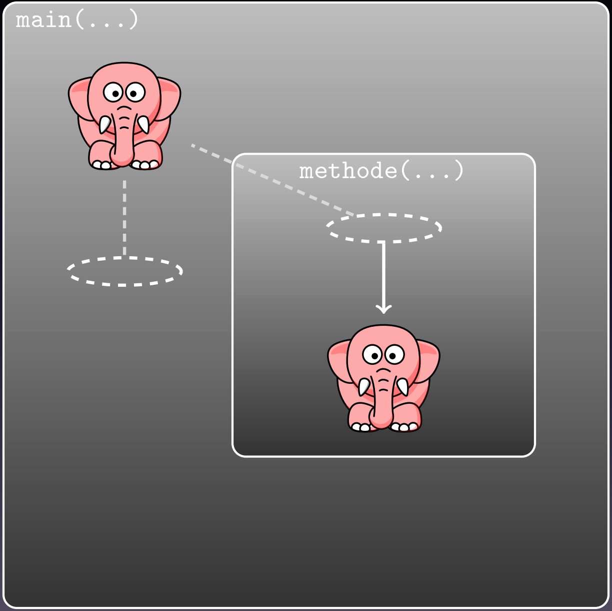
Call by Reference



Call by Reference



Call by Reference



Call by Reference: Beispiel

Call by Reference: Beispiel

```
public class HugeCopy {
    public static void main(String args[]) {
        int [] arr = new int[10000];

        setOne(arr);

    }
    public static void setOne(int arr[]) {
        for(int i=0; i<arr.length; i++) {
            arr[i] = 1;
        }

    }
}
```

Call by Reference: Beispiel

```java
public class HugeCopy {
    public static void main(String args[]) {
        int [] arr = new int[10000];
        System.out.println("before: " + arr[9999]);
        setOne(arr);
        System.out.println("after: " + arr[9999]);
    }
    public static void setOne(int arr[]) {
        for(int i=0; i<arr.length; i++) {
            arr[i] = 1;
        }
        System.out.println("in setOne: " + arr[9999]);
    }
}
```

Call by Reference: Beispiel

```java
public class HugeCopy {
    public static void main(String args[]) {
        int [] arr = new int[10000];
        System.out.println("before: " + arr[9999]);
        setOne(arr);
        System.out.println("after: " + arr[9999]);
    }
    public static void setOne(int arr[]) {
        for(int i=0; i<arr.length; i++) {
            arr[i] = 1;
        }
        System.out.println("in setOne: " + arr[9999]);
    }
}
```

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

Call by Reference: Beispiel

```java
public class HugeCopy {
    public static void main(String args[]) {
        int [] arr = new int[10000];
        System.out.println("before: " + arr[9999]);
        setOne(arr);
        System.out.println("after: " + arr[9999]);
    }
    public static void setOne(int arr[]) {
        for(int i=0; i<arr.length; i++) {
            arr[i] = 1;
        }
        System.out.println("in setOne: " + arr[9999]);
    }
}
```

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

Call by Reference: Beispiel

```java
public class HugeCopy {
    public static void main(String args[]) {
        int [] arr = new int[10000];
        System.out.println("before: " + arr[9999]);
        setOne(arr);
        System.out.println("after: " + arr[9999]);
    }
    public static void setOne(int arr[]) {
        for(int i=0; i<arr.length; i++) {
            arr[i] = 1;
        }
        System.out.println("in setOne: " + arr[9999]);
    }
}
```

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

Call by Reference: Beispiel

```java
public class HugeCopy {
    public static void main(String args[]) {
        int [] arr = new int[10000];
        System.out.println("before: " + arr[9999]);
        setOne(arr);
        System.out.println("after: " + arr[9999]);
    }
    public static void setOne(int arr[]) {
        for(int i=0; i<arr.length; i++) {
            arr[i] = 1;
        }
        System.out.println("in setOne: " + arr[9999]);
    }
}
```

```
~ $ java HugeCopy
before: 0
in setOne: 1
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

Seiteneffekte

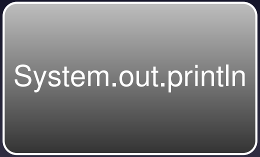
Seiteneffekte

oder:

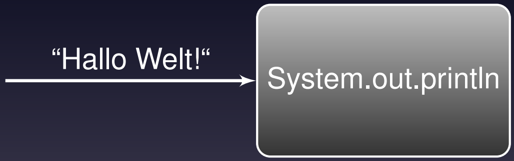
Warum die Black-Box
doch keine Black-Box ist. . .

Seiteneffekte bei System.out.println

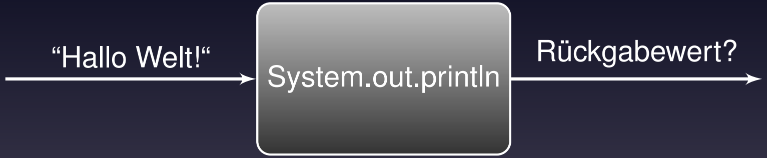
Seiteneffekte bei System.out.println



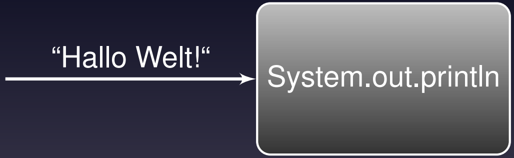
Seiteneffekte bei System.out.println



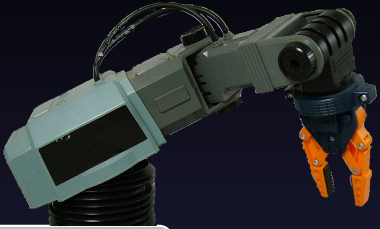
Seiteneffekte bei System.out.println

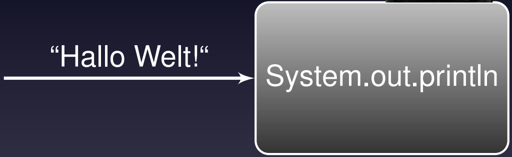


Seiteneffekte bei System.out.println

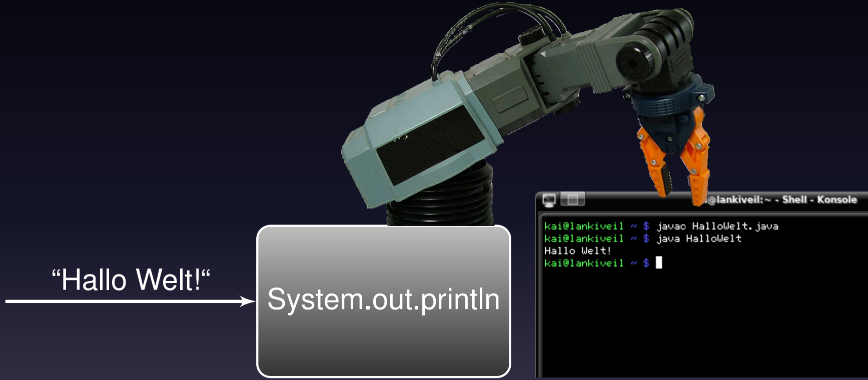


Seiteneffekte bei System.out.println



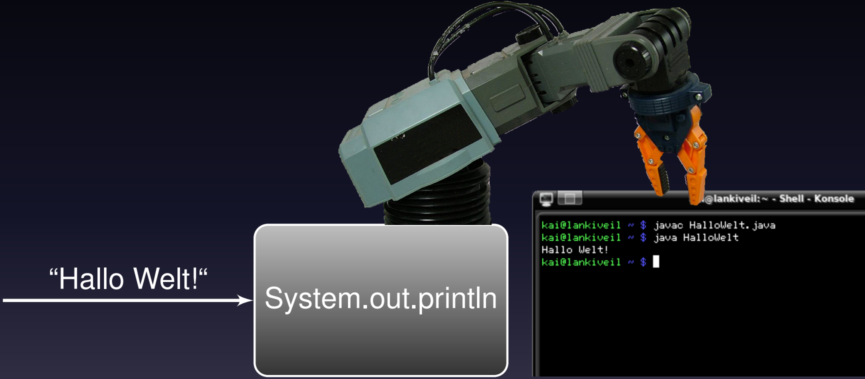


Seiteneffekte bei System.out.println



```
kai@lankiveil ~ $ javac HalloWelt.java
kai@lankiveil ~ $ java HalloWelt
Hallo Welt!
kai@lankiveil ~ $ 
```

Seiteneffekte bei System.out.println



```
kai@lankiveil ~ $ javac HalloWelt.java
kai@lankiveil ~ $ java HalloWelt
Hallo Welt!
kai@lankiveil ~ $ 
```

Methoden haben (leider)
sehr oft Seiteneffekte

2. Testen

Was heißt Testen?

Was kann man Testen?

Was kann man Testen?

Methoden

Wie Testen?

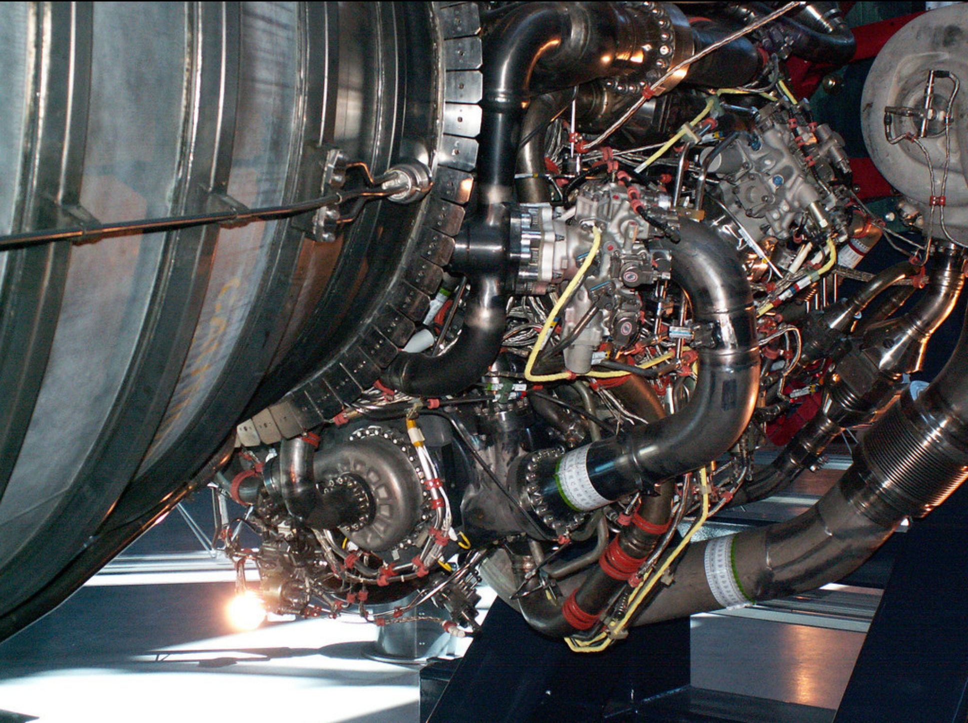
Der Idealfall:

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

Warum Testen?

NASA
Discovery

TLE LAUNCH
ON 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

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

Wie sollte ein Test aussehen?

Factorial.java

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

public static void testFactorial() {



}

public static void main(String args[]) {
    testFactorial();
}
```

Wie sollte ein Test aussehen?

Factorial.java

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

public static void testFactorial() {



}

public static void main(String args[]) {
    testFactorial();
}
```

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

Wie sollte ein Test aussehen?

Factorial.java

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

public static void testFactorial() {
    printTest("factorial", 4, factorial(4), 24);
    printTest("factorial", 1, factorial(1), 1);
    printTest("factorial", 0, factorial(0), 1);
}

public static void main(String args[]) {
    testFactorial();
}
```

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

printTest

```java
public static void printTest(
    String methodName,
    int param,
    int result,
    int expected) {

    System.out.println(
        methodName +
        "(" + param + ")" +
        " expected: " + expected +
        " result: " + result
    );
}
```

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

Factorial implementiert, 1. Versuch

```
public static int factorial(int n) {
        int fac = 1;
        while(n != 0) {
                fac = fac * n;
                n = n - 1;
        }
        return fac;
}
```

Factorial, 1. Versuch, Test

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

Factorial: mehr Tests

```java
public static void testFactorial() {
    printTest("factorial", 4, factorial(4), 24);
    printTest("factorial", 1, factorial(1), 1);
    printTest("factorial", 0, factorial(0), 1);
    printTest("factorial", -1, factorial(-1), 0);
}
```

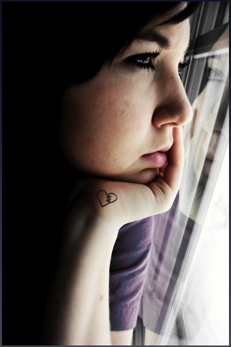
Was passiert?

Factorial, Test

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

Factorial, Test

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



. . . Stunden später . . .

$-1!$

Factorial implementiert, 2. Versuch

```
public static int factorial(int n) {
        if(n<0){return 0;}
        int fac = 1;
        while(n != 0) {
                fac = fac * n;
                n = n - 1;
        }
        return fac;
}
```

Factorial implementiert, 2. Versuch

```java
public static int factorial(int n) {
        if(n<0){return 0;}
        int fac = 1;
        while(n != 0) {
                fac = fac * n;
                n = n - 1;
        }
        return fac;
}
```

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

3. Java-API

Java-API

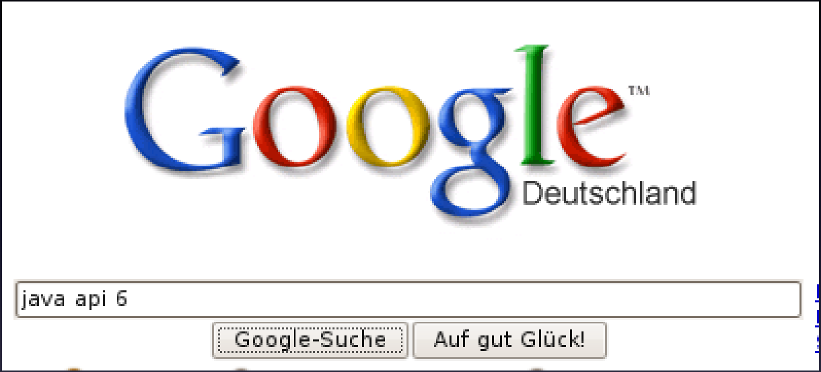
Java-API

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

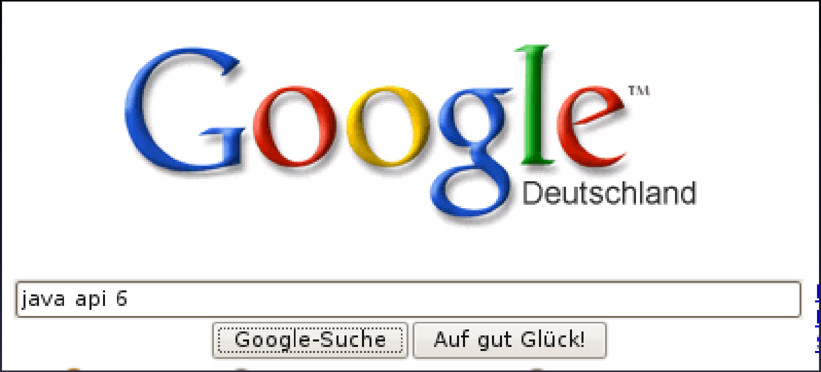
Java-API

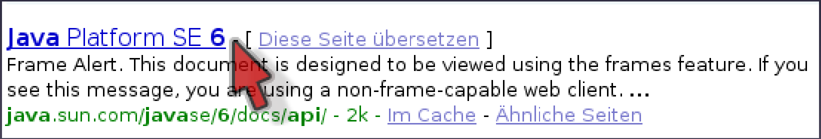
Wie finde ich diese Standard-Funktionen?

Java-API



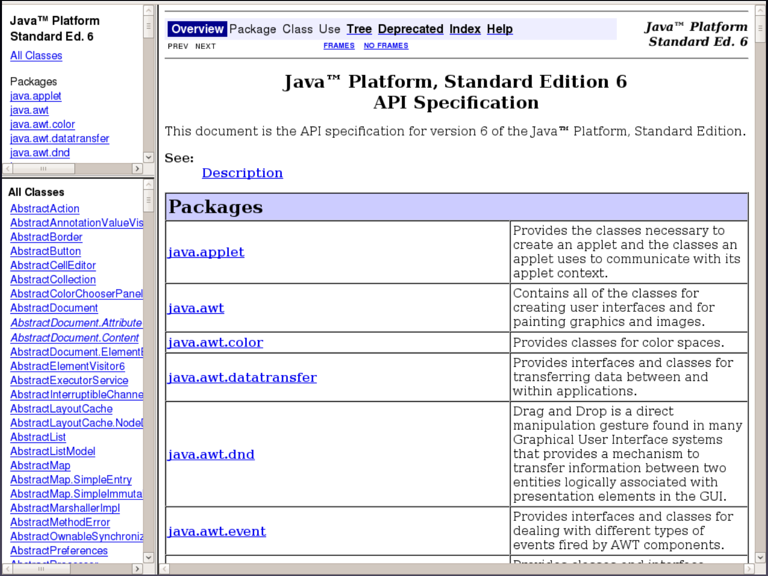
Java-API





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 Ed. 6

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

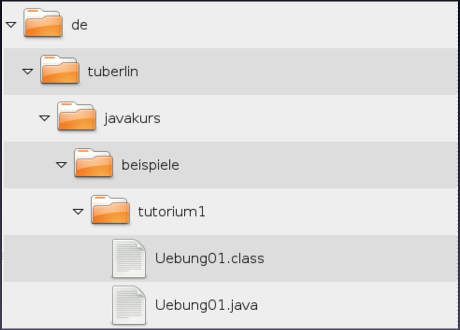
- Ähnlich einer Verzeichnisstruktur

Exkurs: Package

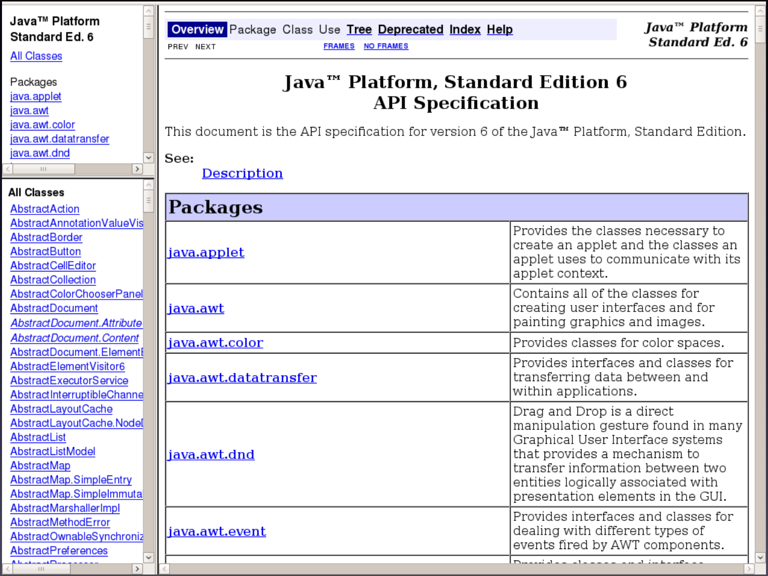
- Ähnlich einer Verzeichnisstruktur
- Strukturierung nach unterschiedlichen Gesichtspunkten,

Exkurs: Package

```java
package de.tuberlin.javakurs.beispiele.tutorium1;

public class Uebung01 {
...
}
```


de
tuberlin
javakurs
beispiele
tutorium1
Uebung01.class
Uebung01.java

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 Ed. 6

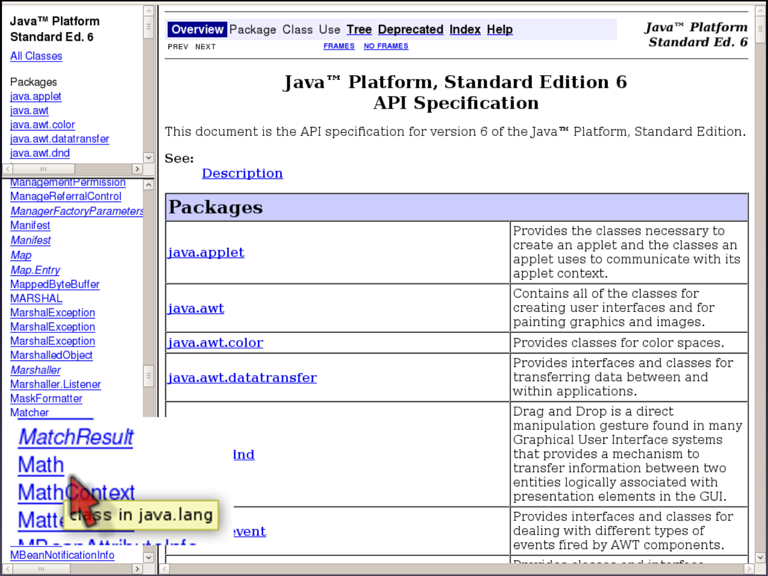
Java™ Platform, Standard Edition 6
API Specification

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

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
Matte
MBeanNotificationInfo

class in java.lang

Overview Package Class Use Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES

Java™ Platform Standard Ed. 6

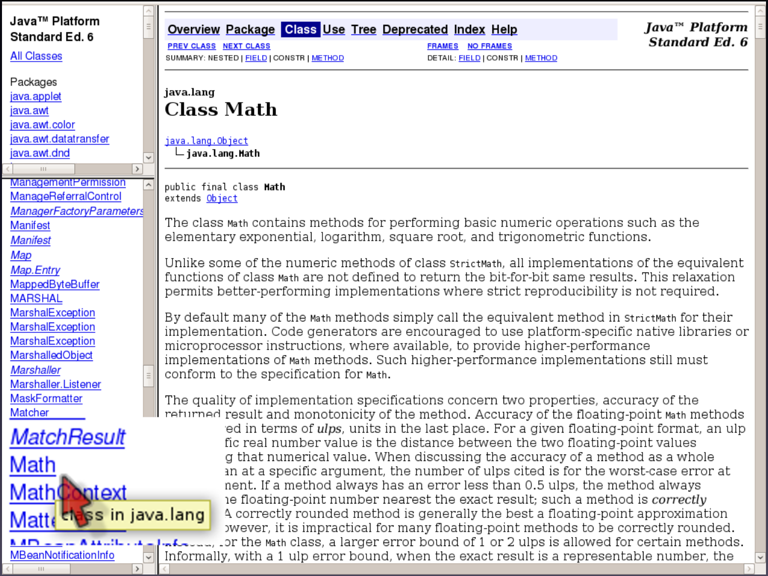
Java™ Platform, Standard Edition 6
API Specification

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.
...nd	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.
...vent	Provides interfaces and classes for dealing with different types of events fired by AWT components.

Java-API - Math.random()



java.lang
Class Math

```
java.lang.Object
  └ java.lang.Math
```

```
public final class Math
extends Object
```

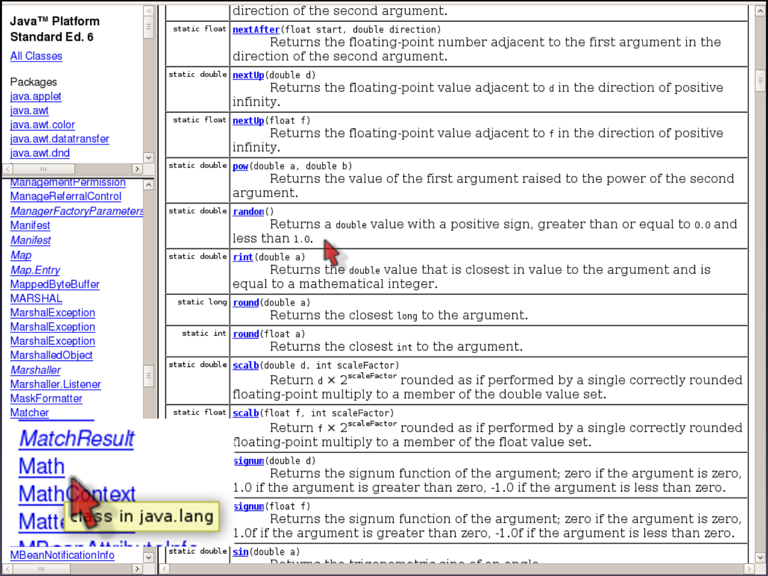
The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

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 ed in terms of *ulps*, units in the last place. For a given floating-point format, an ulp fic 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 an at a specific argument, the number of ulps cited is for the worst-case error at nent. If a method always has an error less than 0.5 ulps, the method always ie floating-point number nearest the exact result; such a method is *correctly* A correctly rounded method is generally the best a floating-point approximation owever, it is impractical for many floating-point methods to be correctly rounded. or 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-API - Math.random()



Java™ Platform
Standard Ed. 6

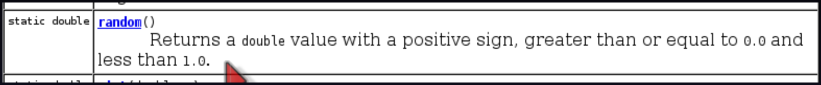
All Classes

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
Matte
class in java.lang
MBeanNotificationInfo

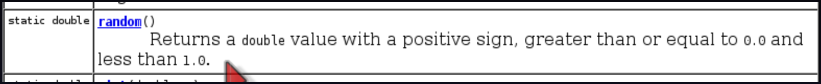
	direction of the second argument.
static float	**nextAfter**(float start, double direction) Returns the floating-point number adjacent to the first argument in the direction of the second argument.
static double	**nextUp**(double d) Returns the floating-point value adjacent to d in the direction of positive 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 × $2^{\text{scaleFactor}}$ 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 × $2^{\text{scaleFactor}}$ 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)

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.

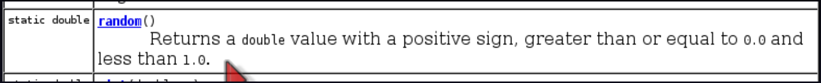
Java-API - Math.random()



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

- Bezeichnung

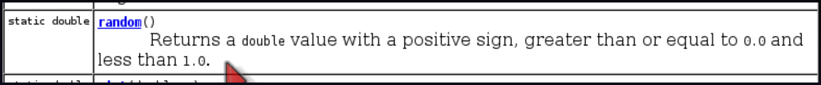
Java-API - Math.random()



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

- Bezeichnung
- Beschreibung

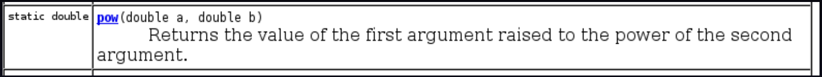
Java-API - Math.random()



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

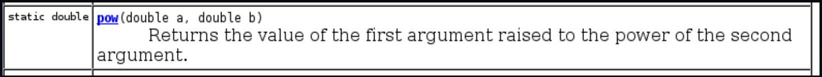
- Bezeichnung
- Beschreibung
- Rückgabewert und Typ

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

4. Namensgebung

Namensgebung

Variablen, Parameter, Methoden, Klassen und Packages

Namensgebung

Variablen, Parameter, Methoden, Klassen und Packages

- identifizieren ein Programmierobjekt

Namensgebung

Variablen, Parameter, Methoden, Klassen und Packages

- identifizieren ein Programmierobjekt
- können den Inhalt bzw. die Nutzung beschreiben

Namensgebung

Variablen, Parameter, Methoden, Klassen und Packages

- identifizieren ein Programmierobjekt
- können den Inhalt bzw. die Nutzung beschreiben

Grundlage der Kommunikation:

Namensgebung

Variablen, Parameter, Methoden, Klassen und Packages

- identifizieren ein Programmierobjekt
- können den Inhalt bzw. die Nutzung beschreiben

Grundlage der Kommunikation:

- Entwickler
- Teampartner
- Tutoren
- ...

Namensgebung - Beispiel

```
int z = 50;
if ( a(z) ) {
        z = b();
} else {
        z = c(z);
}
```

Namensgebung - Beispiel

```
int z = 50;
if ( a(z) ) {
        z = b();
} else {
        z = c(z);
}

int hoehe = 50;
if ( istDerRasenZuHoch(hoehe) ) {
  hoehe = maeheDenRasen();
} else {
  hoehe = legeDichInDieSonne(hoehe);
}
```

Namensgebung - Beispiel

```
int z = 50;
if ( a(z) ) {
        z = b();
} else {
        z = c(z);
}

int hoehe = 50;
if ( istDerRasenZuHoch(hoehe) ) {
  hoehe = maeheDenRasen();
} else {
  hoehe = lassRasenWachsen(hoehe);
}
```

Namensgebung - Beispiel

```
int z = 50;
if ( a(z) ) {
        z = b();
} else {
        z = c(z);
}

int hoehe = 50;
if ( istDerRasenZuHoch(hoehe) ) {
  hoehe = maeheDenRasen();
} else {
  hoehe = lassRasenWachsen(hoehe);
}
```

- ebenfalls Namen der Parameter, Variablen, Klassen und Packages

Namensgebung - Don’ts

Namensgebung - Don’ts

- extrem lange Namen

Namensgebung - Don’ts

- extrem lange Namen
- allgemeine Namen

Namensgebung - Don’ts

- extrem lange Namen
- allgemeine Namen
- unzutreffende bzw. irreführende Namen

Namensgebung - Do’s

Namensgebung - Do’s

- aussagekräftig Namen

Namensgebung - Do’s

- aussagekräftig Namen
- kurz, knapp, präzise

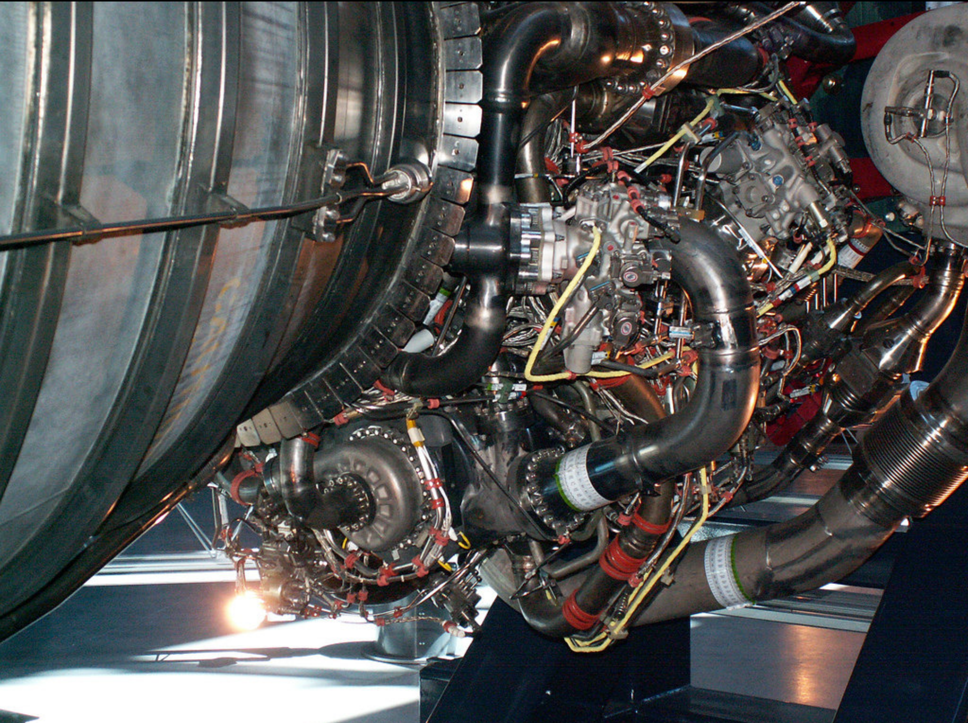
Namensgebung - Do’s

- aussagekräftig Namen
- kurz, knapp, präzise

Beispielsweise:

- fakultaet(int n)
- maeheDenRasen(int hoehe)
- ...

5. Debugging



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

Systematik

- Fehlerstelle eingrenzen
- Programmablauf überprüfen

Systematik

- Fehlerstelle eingrenzen
- Programmablauf überprüfen

- durch: Kontrollausgaben

Beispiel - Modulo

```java
public static int modulo(int zahl, int divisor) {
  int modulo = zahl;

  while(modulo > divisor) {


    modulo = modulo - divisor;
  }

  return modulo;
}
```

Beispiel - Modulo

Code wurde nicht getestet

Beispiel - Modulo

Code wurde nicht getestet

BANG!

ohne Kontrollausgaben

```
public static int modulo(int zahl, int divisor) {
  int modulo = zahl;

  while(modulo > divisor) {


    modulo = modulo - divisor;
  }

  return modulo;
}
```

mit Kontrollausgaben

```
public static int modulo(int zahl, int divisor) {
  int modulo = zahl;
  System.out.println(zahl + "%" + divisor);
  while(modulo > divisor) {
    System.out.print("modulo - divisor: " + modulo +
        "- " divisor + " = " + (modulo -divisor));
    modulo = modulo - divisor;
  }
  System.out.println(zahl+"%"+divisor+"=" + modulo);
  return modulo;
}
```

Ausgaben

Ausgaben

```
7%0
modulo - divisor: 7 - 0 = 7
modulo - divisor: 7 - 0 = 7
modulo - divisor: 7 - 0 = 7
modulo - divisor: 7 - 0 = 7
modulo - divisor: 7 - 0 = 7
modulo - divisor: 7 - 0 = 7
...
STRG-C
```

Debugging - 7%0

```
public static int modulo(int zahl, int divisor) {
  int modulo = zahl;
  //System.out.println(zahl + "%" + divisor);
  while(modulo > divisor) {
    //System.out.print("modulo - divisor: " + modulo +
    //  "- " divisor + " = " + (modulo -divisor));
    modulo = modulo - divisor;
  }
  //System.out.println(zahl+"%"+divisor+"=" + modulo);
  return modulo;
}
```

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/