

# Hello World

## 1. Vorlesung, Javakurs 2010

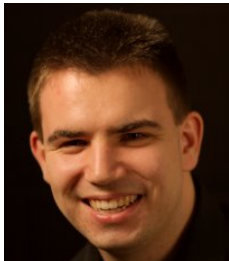
Mario Bodemann    Sebastian Dyroff

[www.freitagrunde.org](http://www.freitagrunde.org)

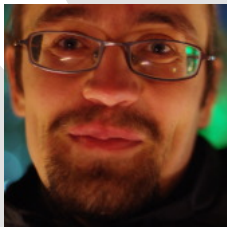
21. März 2010



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



Sebastian



Mario

- ▶ Studierende wie Ihr, Mitglieder der Freitagsrunde
- ▶ Informatik
- ▶ Diplom



- ▶ Studentische Initiative voller Studenten
- ▶ Gesamte Fak. IV: ET, TI, Info
- ▶ Organisiert Kurse, Kickerturniere, Gremienarbeit, Beamerverleihe, Keysignings, Einführungswochen, Klausurensammlungen, ...
- ▶ [www.freitagsrunde.org](http://www.freitagsrunde.org)
- ▶ Freitags, 13 Uhr im FR 5046

# Inhaltsverzeichnis

- 1 Organisatorisches
- 2 Kompilieren und Ausführen
- 3 Hello World - das erste Programm
  - Ausführen
- 4 Abarbeiten von Befehlen
- 5 Variablen und einfache Typen
- 6 Operatoren
- 7 Fallunterscheidungen
- 8 Kommentare
- 9 Probleme und Fehlermeldungen
- 10 Zusammenfassung



4!

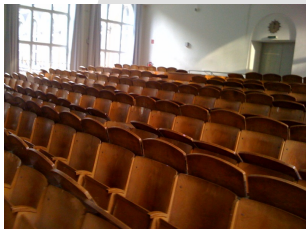
# Inhaltsverzeichnis

- 1** Organisatorisches
- 2 Kompilieren und Ausführen
- 3 Hello World - das erste Programm
  - Ausführen
- 4 Abarbeiten von Befehlen
- 5 Variablen und einfache Typen
- 6 Operatoren
- 7 Fallunterscheidungen
- 8 Kommentare
- 9 Probleme und Fehlermeldungen
- 10 Zusammenfassung



4!

	Mo	Di	Mi	Do
10:15	<i>Hello World</i>	<i>Methoden</i>	<i>Aufgaben</i>	<i>OOP</i>
11:30	Übung	Übung	Übung	Übung
13:15	Mittag	Mittag	Mittag	Mittag
14:15	<i>Schleifen</i>	Übung	<i>Kapselung</i>	Übung
15:30	Übung		Übung	



H2032



TEL 106 / TEL 206

# Ziele des Kurses

- ▶ Für euch:
  - ▷ Javaprogramme schreiben
  - ▷ Grundlagen von Java erlernen
  - ▷ Probleme im Code finden und lösen
- ▶ Für uns:
  - ▷ Üben von Vorträgen
  - ▷ Lernen zu Sprechen
  - ▷ Resonanz zum Vortragsstil
- ▶ Zusammen: Viel, viel Spaß haben ...



3

# Inhaltsverzeichnis

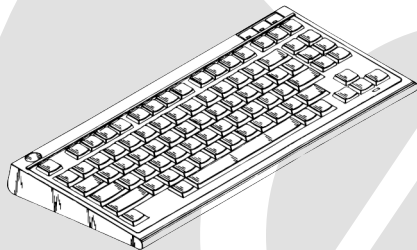
- 
- 1 Organisatorisches
  - 2 Kompilieren und Ausführen
  - 3 Hello World - das erste Programm
    - Ausführen
  - 4 Abarbeiten von Befehlen
  - 5 Variablen und einfache Typen
  - 6 Operatoren
  - 7 Fallunterscheidungen
  - 8 Kommentare
  - 9 Probleme und Fehlermeldungen
  - 10 Zusammenfassung



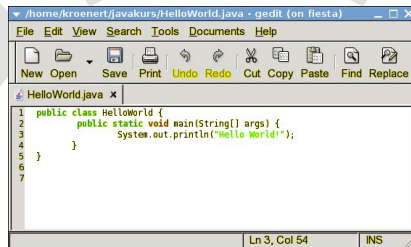


*Arbeitsumgebung*

## ► Einloggen



- ▶ Einloggen
- ▶ einen Editor <sup>4</sup> öffnen



The screenshot shows the Gedit text editor window. The title bar reads "/home/kroenert/javakurs/HelloWorld.java - gedit (on fiesta)". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The text area shows the following code:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println("Hello World!");  
4     }  
5 }  
6  
7
```

The status bar at the bottom indicates "Ln 3, Col 54" and "INS" mode.

z.b. **Gedit** (Gnome) oder **kate** (KDE)

---

<sup>4</sup>Nein! Nicht Word!

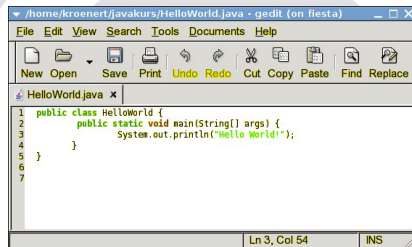
# Hello World

Listing 1: src/HelloWorld.java

```
1 public class HelloWorld {  
2  
3     public static void main(String[] args) {  
4  
5         System.out.println("Hello World!");  
6  
7     }  
8 }
```

Beim starten des Programs wird die “main” Methode ausgeführt

- ▶ Einloggen
- ▶ einen Editor öffnen
- ▶ Programm schreiben



The screenshot shows a text editor window titled "/home/kroenert/javakurs/HelloWorld.java - gedit (on fiesta)". The window has a menu bar with "File", "Edit", "View", "Search", "Tools", "Documents", and "Help". Below the menu bar is a toolbar with icons for New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main text area contains the following Java code:

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println("Hello World!");  
4     }  
5 }  
6  
7
```

The status bar at the bottom right shows "Ln 3, Col 54" and "INS".

- ▶ Einloggen
- ▶ einen Editor öffnen
- ▶ Programm schreiben
- ▶ eine Shell <sup>5</sup> öffnen



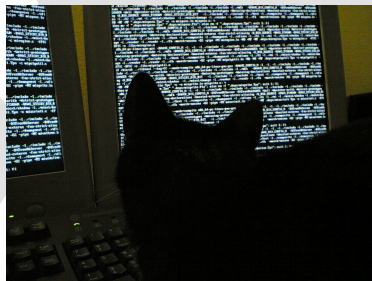
---

<sup>5</sup>Konsole, Terminal, (MS-DOS-)Eingabeaufforderung, Kommandozeile

# Kompilieren

Der Compiler übersetzt den Quellcode in ein ausführbares Programm.

**javac** ist der **Java** Compiler.



```
$ javac HelloWorld.java  
$
```

Kompilieren eines Java Programms

```
$ ls -l
-rw----- 1 dyroff 426 Mar 22 10:30 HelloWorld.class
-rw----- 1 dyroff 106 Mar 22 10:29 HelloWorld.java
$
```

## Ausgabedatei des Javakompilers

- ▶ Kompilieren erzeugt .class Dateien, sog. Bytecode
- ▶ Bytecode kann mit einer **Java Virtual Machine** ausgeführt werden
- ▶ Bytecode ist maschinenunabhängig

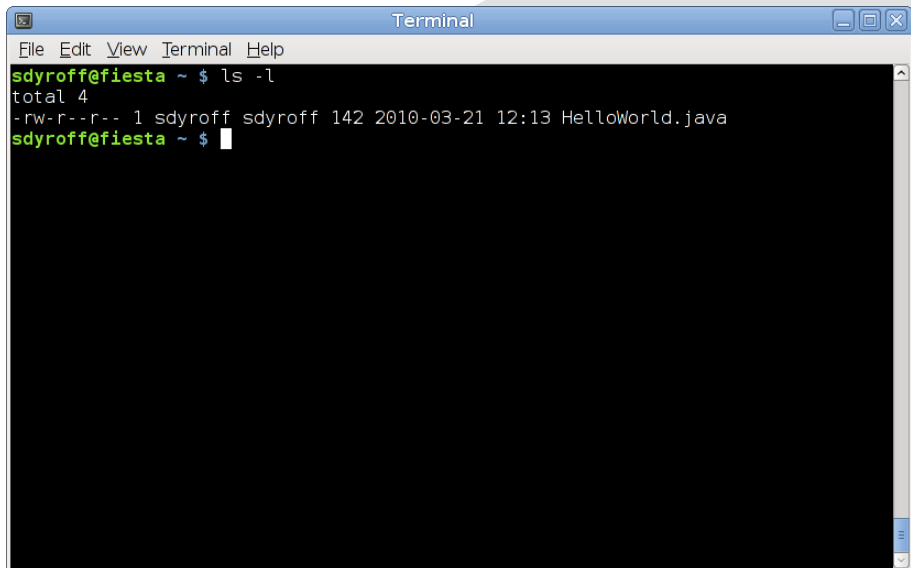


```
$ java HelloWorld  
Hello World!  
$
```

## Ausgabe unseres Programms

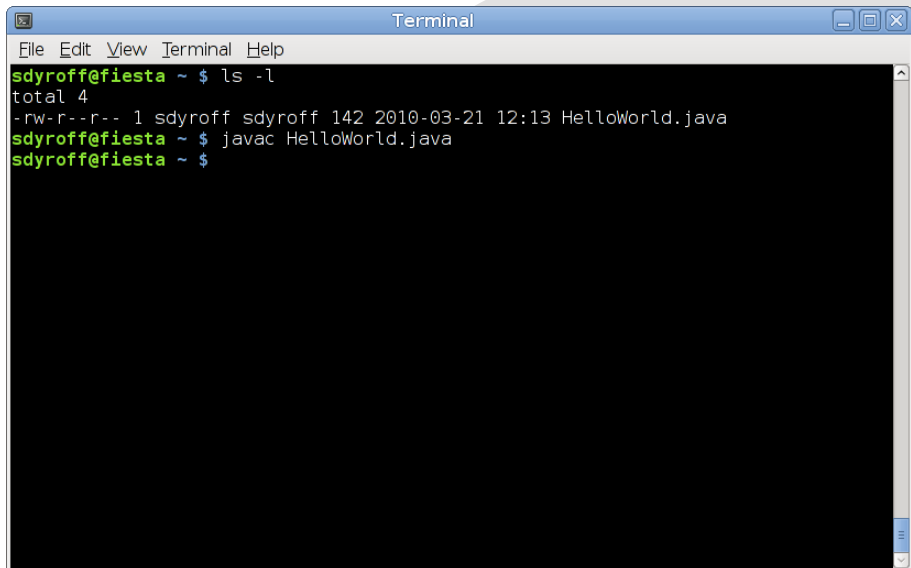
- ▶ **java** ist die Java Virtual Maschine
- ▶ als Parameter wird der Klassenname übergeben
- ▶ die Ausgabe ist auf der Console zu sehen

4!



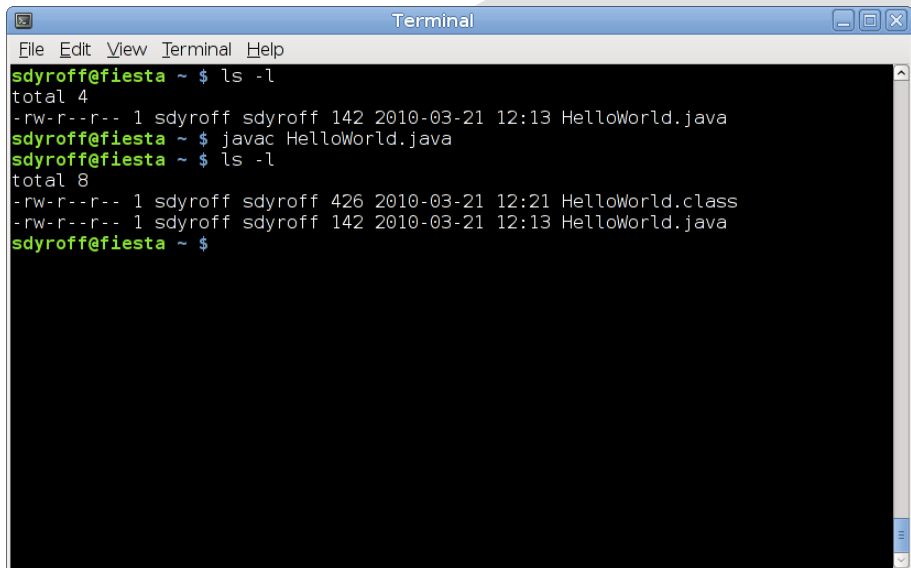
A screenshot of a macOS Terminal window titled "Terminal". The window has a blue title bar with standard macOS window controls (minimize, maximize, close) on the right. Below the title bar is a menu bar with the following items: File, Edit, View, Terminal, and Help. The main area of the window is black with white text. The prompt is "sdyroff@fiesta ~ \$". The command entered is "ls -l". The output is "total 4" followed by a single line: "-rw-r--r-- 1 sdyroff sdyroff 142 2010-03-21 12:13 HelloWorld.java". The prompt "sdyroff@fiesta ~ \$" is shown again with a white cursor. On the right side of the terminal area, there are vertical scroll bars with up and down arrows.

```
sdyroff@fiesta ~ $ ls -l
total 4
-rw-r--r-- 1 sdyroff sdyroff 142 2010-03-21 12:13 HelloWorld.java
sdyroff@fiesta ~ $
```

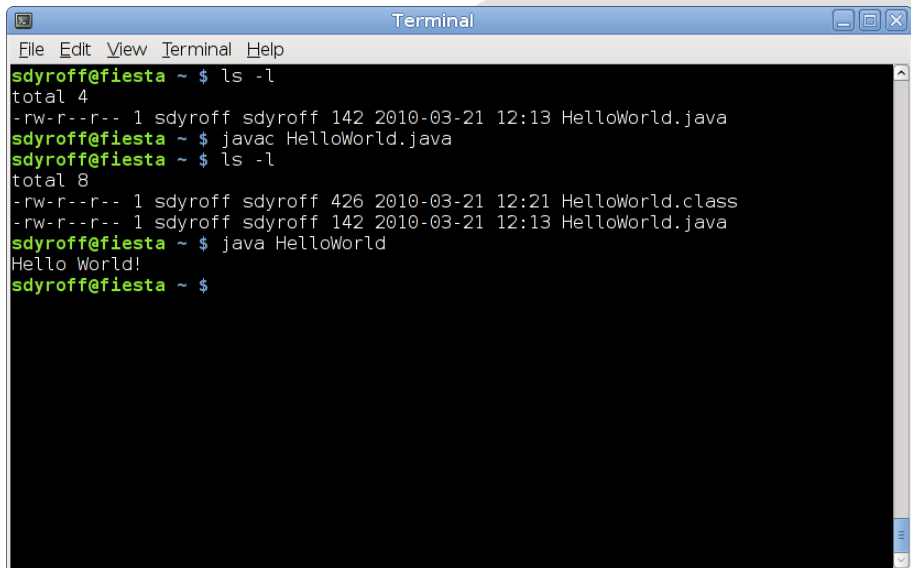


A screenshot of a macOS Terminal window titled "Terminal". The window has a blue title bar with standard window controls (minimize, maximize, close) on the right. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", and "Help" menus. The main area is a black terminal with green text for the prompt and white text for the output. The user is at the "fiesta" machine, indicated by the prompt "sdyroff@fiesta ~". The first command is "ls -l", which lists the file "HelloWorld.java" with permissions "-rw-r--r--", owner "sdyroff", group "sdyroff", size "142", and timestamp "2010-03-21 12:13". The second command is "javac HelloWorld.java", which compiles the file. The third line shows the prompt again, indicating the command has completed.

```
sdyroff@fiesta ~ $ ls -l
total 4
-rw-r--r-- 1 sdyroff sdyroff 142 2010-03-21 12:13 HelloWorld.java
sdyroff@fiesta ~ $ javac HelloWorld.java
sdyroff@fiesta ~ $
```

A screenshot of a macOS Terminal window titled "Terminal". The window has a blue title bar with standard window controls (minimize, maximize, close) on the right. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", and "Help". The main area is a black terminal with green text. The user "sdyroff" is at the "fiesta" machine in the home directory (~). They run "ls -l", showing a file "HelloWorld.java" with permissions "-rw-r--r--", size 142, and timestamp "2010-03-21 12:13". Then they run "javac HelloWorld.java". Finally, they run "ls -l" again, showing two files: "HelloWorld.class" (size 426, timestamp "2010-03-21 12:21") and "HelloWorld.java" (size 142, timestamp "2010-03-21 12:13"). The prompt "sdyroff@fiesta ~ \$" is shown at the end.

```
sdyroff@fiesta ~ $ ls -l
total 4
-rw-r--r-- 1 sdyroff sdyroff 142 2010-03-21 12:13 HelloWorld.java
sdyroff@fiesta ~ $ javac HelloWorld.java
sdyroff@fiesta ~ $ ls -l
total 8
-rw-r--r-- 1 sdyroff sdyroff 426 2010-03-21 12:21 HelloWorld.class
-rw-r--r-- 1 sdyroff sdyroff 142 2010-03-21 12:13 HelloWorld.java
sdyroff@fiesta ~ $
```

A screenshot of a macOS Terminal window titled "Terminal". The window has a blue title bar with standard window controls (minimize, maximize, close) on the right. Below the title bar is a menu bar with "File", "Edit", "View", "Terminal", and "Help". The main area is a black terminal with green text for the prompt and white text for the output. The user is at a shell prompt "sdyroff@fiesta ~ \$". They run "ls -l", which shows a file "HelloWorld.java" with permissions "-rw-r--r--", owner "sdyroff", group "sdyroff", size "142", and timestamp "2010-03-21 12:13". Then they run "javac HelloWorld.java". They run "ls -l" again, showing two files: "HelloWorld.class" (size "426", timestamp "2010-03-21 12:21") and "HelloWorld.java" (size "142", timestamp "2010-03-21 12:13"). Finally, they run "java HelloWorld", which outputs "Hello World!". The prompt "sdyroff@fiesta ~ \$" is shown again at the bottom. A vertical scrollbar is on the right side of the terminal area.

```
sdyroff@fiesta ~ $ ls -l
total 4
-rw-r--r-- 1 sdyroff sdyroff 142 2010-03-21 12:13 HelloWorld.java
sdyroff@fiesta ~ $ javac HelloWorld.java
sdyroff@fiesta ~ $ ls -l
total 8
-rw-r--r-- 1 sdyroff sdyroff 426 2010-03-21 12:21 HelloWorld.class
-rw-r--r-- 1 sdyroff sdyroff 142 2010-03-21 12:13 HelloWorld.java
sdyroff@fiesta ~ $ java HelloWorld
Hello World!
sdyroff@fiesta ~ $
```

Listing 5: src/HelloSolarSystem.java

```
1 public class HelloSolarSystem {  
2  
3     public static void main(String[] args) {  
4  
5         System.out.println(" Hello_Mercury!");  
6         System.out.println(" Hello_Venus!");  
7         System.out.println(" Hello_Earth!");  
8         System.out.println(" Hello_Mars!");  
9         System.out.println(" Hello_Jupiter!");  
10        System.out.println(" Hello_Saturn!");  
11        System.out.println(" Hello_Uranus!");  
12        System.out.println(" Hello_Neptune!");  
13  
14    }  
15  
16 }
```

# Abarbeiten von Befehlen

Listing 6: src/HelloSolarSystem.java

```
1 public class HelloSolarSystem {  
2  
3     public static void main(String[] args) {  
4  
5         System.out.println("Hello_Mercury!");  
6         System.out.println("Hello_Venus!");  
7         System.out.println("Hello_Earth!");  
8         System.out.println("Hello_Mars!");  
9         System.out.println("Hello_Jupiter!");  
10        System.out.println("Hello_Saturn!");  
11        System.out.println("Hello_Uranus!");  
12        System.out.println("Hello_Neptune!");  
13    }  
14 }  
15  
16 }
```

- ▶ Befehle werden der Reihe nach abgearbeitet
- ▶ Klassennamen und Dateiname (ohne .java) müssen übereinstimmen

# Inhaltsverzeichnis

- 
- 1 Organisatorisches
  - 2 Kompilieren und Ausführen
  - 3 Hello World - das erste Programm
    - Ausführen
  - 4 Abarbeiten von Befehlen
  - 5 Variablen und einfache Typen**
  - 6 Operatoren
  - 7 Fallunterscheidungen
  - 8 Kommentare
  - 9 Probleme und Fehlermeldungen
  - 10 Zusammenfassung



Listing 7: src/Variables.java

```
1 public class Variables {  
2     public static void main(String[] args) {  
3  
4         // Deklaration einer Variablen  
5         int number;  
6         // Initialisierung einer Variablen  
7         number = 23;  
8         System.out.println(number);  
9  
10    }  
11 }
```

- ▶ **int** steht für **Integer**, eine ganze Zahl
- ▶ **=** weist den rechten Wert der Variablen auf der Linken zu

```
$ javac Variables.java  
$ java Variables  
23  
$
```

Ausgabe Variables

- ▶ Kompilieren und Ausführen
- ▶ Der Wert der Variablen wird auf die Konsole geschrieben

Listing 9: src/VariablesII.java

```
1 public class VariablesII {  
2     public static void main(String [] args) {  
3  
4         int age = 20;  
5         int number = 3;  
6  
7         age = age + number;  
8         String message;  
9         message = "My_age_is: ";  
10  
11  
12         System.out.println(message + age);  
13  
14     }  
15 }  
16 }
```

```
$ javac VariablesII.java  
$ java VariablesII  
My age is: 23  
$
```

Ausgabe VariablesII

4!

Listing 11: src/VariablesIII.java

```
1 public class VariablesIII {  
2     public static void main(String [] args) {  
3  
4         double height = 1.75;  
5         String message = "My_height_is_";  
6         System.out.println(message + height);  
7  
8     }  
9 }
```

► **double** ist eine Fließkommazahl

```
$ javac VariablesIII.java  
$ java VariablesIII  
My height is 1.75  
$
```

Ausgabe VariablesIII

4!

Listing 13: src/VariablesIV.java

```
1 public class VariablesIV {  
2     public static void main(String[] args) {  
3  
4         boolean amlSmart = true;  
5         boolean amlAJavaHacker = false;  
6  
7         boolean result = amlSmart && amlAJavaHacker;  
8  
9         String message = "Am_I_a_smart_Java_hacker?";  
10        System.out.println(message + result);  
11    }  
12 }
```

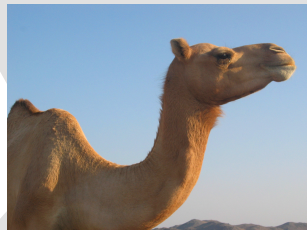
```
$ javac VariablesIV.java  
$ java VariablesIV  
Am I a smart Java hacker? false  
$
```

Ausgabe VariablesIV



# Konventionen

- ▶ Variablennamen werden im sogenannten camelCase geschrieben
- ▶ Der erste Buchstabe ist aber klein.



Zum Beispiel: **mySolarSystem**

Empfehlung:

- ▶ kurze und aussagekräftige Namen verwenden

4!

# Datentypen im Überblick

Typ

Wertebereich

## Datentypen

```
1 boolean result = false;  
2 int age = 20;  
3 double height = 1.75;  
4 String message = "Javakurs";
```

{true;false}

{-2147483648 ... 2147483647}

{ $\pm 4,9 \cdot 10^{-324}$  ...  $\pm 1,7977 \cdot 10^{+308}$ }

{ "... endlich }

Es gibt zwar noch mehr Datentypen, aber dies sind erstmal die wichtigsten.

# Inhaltsverzeichnis

- 
- 1 Organisatorisches
  - 2 Kompilieren und Ausführen
  - 3 Hello World - das erste Programm
    - Ausführen
  - 4 Abarbeiten von Befehlen
  - 5 Variablen und einfache Typen
  - 6 Operatoren**
  - 7 Fallunterscheidungen
  - 8 Kommentare
  - 9 Probleme und Fehlermeldungen
  - 10 Zusammenfassung

Listing 16: src/Operators.java

```
1 public class Operators {  
2     public static void main(String [] args) {  
3  
4         int a, b;  
5         a = 10;  
6         b = 2;  
7  
8         int multi = a * b;  
9         int average = (a + b) / 2;  
10  
11     }  
12 }
```

► es gelten die üblichen Rechenregeln

## Logische Operatoren

&&	und
	oder
!	Negation

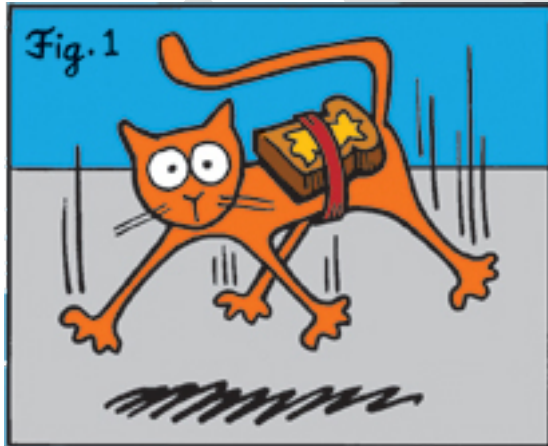
## Arithmetische Operatoren

+	Addition
-	Subtraktion
/	Division
*	Multiplikation
%	Modulo

4!

# Inhaltsverzeichnis

- 
- 1 Organisatorisches
  - 2 Kompilieren und Ausführen
  - 3 Hello World - das erste Programm
    - Ausführen
  - 4 Abarbeiten von Befehlen
  - 5 Variablen und einfache Typen
  - 6 Operatoren
  - 7 Fallunterscheidungen**
  - 8 Kommentare
  - 9 Probleme und Fehlermeldungen
  - 10 Zusammenfassung



Listing 17: src/IfSimple.java

```
4  boolean catDown = true;
5  if( catDown == true ) {
6      System.out.println("Toast_wird_Katze_drehen...");
7  }
8  if( catDown == false ) {
9      System.out.println("Katze_wird_Toast_drehen...");
10 }
```



## Einfaches Beispiel (mit else)

Listing 18: src/IfSimpleCorrect.java

```
4  boolean catDown = true;  
5  if( catDown == true ) {  
6      System.out.println("Toast_wird_Katze_drehen...");  
7  } else {  
8      System.out.println("Katze_wird_Toast_drehen...");  
9  }
```

# Einfaches Beispiel (mit else)

Listing 19: src/IfSimpleCorrect.java

```
4  boolean catDown = true;  
5  if( catDown == true ) {  
6      System.out.println("Toast_wird_Katze_drehen...");  
7  } else {  
8      System.out.println("Katze_wird_Toast_drehen...");  
9  }
```

## Ausgabe

```
[mario src]$ java IfSimpleCorrect  
Toast wird Katze drehen ...
```

# Bedingungen mit logischen Operatoren

Listing 20: src/IfGrades.java

```
4  int points = 77;  
5  if( points <= 50 ) {  
6      System.out.println(" Leider_nicht_bestanden_...");  
7  } else {  
8      System.out.println(" Bestanden_!!");  
9  }
```

# Bedingungen mit logischen Operatoren

Listing 21: src/IfGrades.java

```
4  int points = 77;  
5  if( points <= 50 ) {  
6      System.out.println(" Leider_nicht_bestanden_...");  
7  } else {  
8      System.out.println(" Bestanden_!!");  
9  }
```

## Ausgabe

```
[mario src]$ java IfGrades  
Bestanden !!
```

# Bedingungen kombiniert (mit else if)

Listing 22: src/IfGradesMixedElseIf.java

```
4  boolean testPassed = true;
5  int points = 35;
6
7  if( points > 50 && testPassed ) {
8      System.out.println("Klausur_bestanden!");
9  } else if( testPassed ) {
10     System.out.println("Klausur_wiederholen");
11 } else {
12     System.out.println("Durchgefallen...");
13 }
```

# Bedingungen kombiniert (mit else if)

Listing 23: src/IfGradesMixedElseIf.java

```
4  boolean testPassed = true;
5  int points = 35;

6
7  if( points > 50 && testPassed ) {
8      System.out.println(" Klausur_bestanden!");
9  } else if( testPassed ) {
10     System.out.println(" Klausur_wiederholen");
11 } else {
12     System.out.println(" Durchgefallen...");
13 }
```

## Ausgabe

```
[mario src]$ java IfGradesMixedElseIf
Klausur wiederholen
```

Listing 24: src/IfSwitch.java

```
3  int number = 51231;
4
5  System.out.print("Zahlendet_auf_");
6  switch( number % 10 ) {
7      case 0: System.out.println("Null");
8          break;
9      case 1: System.out.println("Eins");
10         break;
11     // ...
12     default: System.out.println("Fehler!");
13 }
```

# Ganzzahlige Bedingungen

Listing 25: src/IfSwitch.java

```
3  int number = 51231;
4
5  System.out.print("Zahl endet auf ");
6  switch( number % 10 ) {
7      case 0: System.out.println("Null");
8              break;
9      case 1: System.out.println("Eins");
10             break;
11     // ...
12     default: System.out.println("Fehler!");
13 }
```

## Ausgabe

```
[mario src]$ java IfSwitch
Zahl endet auf Eins
```



# Inhaltsverzeichnis

- 
- 1 Organisatorisches
  - 2 Kompilieren und Ausführen
  - 3 Hello World - das erste Programm
    - Ausführen
  - 4 Abarbeiten von Befehlen
  - 5 Variablen und einfache Typen
  - 6 Operatoren
  - 7 Fallunterscheidungen
  - 8 Kommentare**
  - 9 Probleme und Fehlermeldungen
  - 10 Zusammenfassung

- ▶ Warum mache ich das Folgende?
- ▶ Was gibt es für Randbedingungen?
- ▶ Warum sollte ich dies nicht anders lösen?
- ▶ Wichtig: Kommentar vor dem Code!
  - ▷ Hilft beim Denken
  - ▷ Hilft beim Nachvollziehen



4!

# Beweis durch Gegenbeispiel

Listing 26: src/CommentCounterProof.java

```
1 public class CommentCounterProof { public static void  
    main(String argv []) { System.out.println("Not_a_good_  
    Example"); }}
```

- ▶ Kann das jemand lesen?
- ▶ Was bewirken die Variablen?
- ▶ Wie machen dies andere?

4!

# Lesbarer Beweis durch Gegenbeispiel

Listing 27: src/CommentedCounterProof.java

```
1  /*
2   Samplecode for "correct" commenting.
3
4   This code will demonstrate how to use comments,
5   either in a multiline fasion(like this one), or
6   by single line
7  */
8  public class CommentedCounterProof {
9      // main entry point of class
10     public static void main(String argv[]) {
11         // Supply our user with some basic
12         // information about this sample.
13         System.out.println("A_better_Example");
14     }
15 } // done with this sample
```

# Lesbarer Beweis durch Gegenbeispiel


Listing 28: src/CommentedCounterProof.java

```
1  /*
2   Samplecode for "correct" commenting.
3
4   This code will demonstrate how to use comments,
5   either in a multiline fasion(like this one), or
6   by single line
7  */
8  public class CommentedCounterProof {
9      // main entry point of class
10     public static void main(String argv[]) {
11         // Supply our user with some basic
12         // information about this sample.
13         System.out.println("A better_
14     }
15 } // done with this sample
```

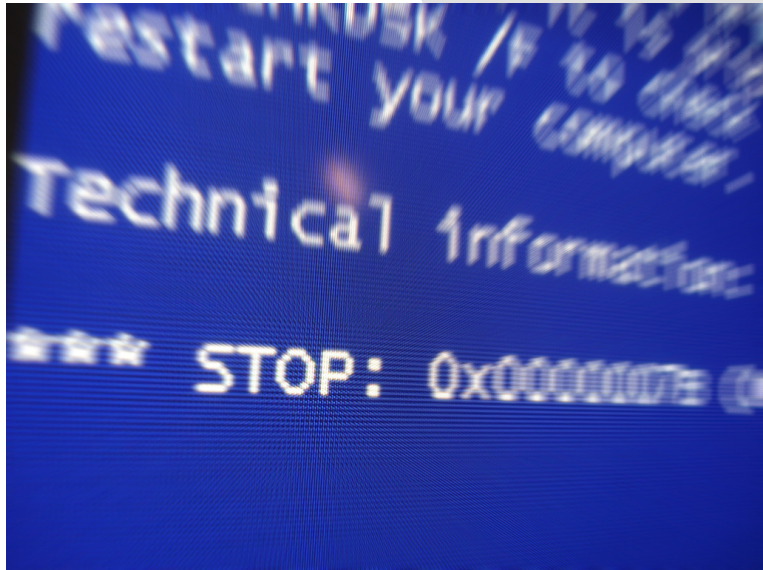
## Erklärung

// Einzeiliger Kommentar  
/\* Kommentar bis \*/

# Inhaltsverzeichnis

- 
- 1 Organisatorisches
  - 2 Kompilieren und Ausführen
  - 3 Hello World - das erste Programm
    - Ausführen
  - 4 Abarbeiten von Befehlen
  - 5 Variablen und einfache Typen
  - 6 Operatoren
  - 7 Fallunterscheidungen
  - 8 Kommentare
  - 9 Probleme und Fehlermeldungen**
  - 10 Zusammenfassung

4!



Listing 29: src/no-compile/ErrorOne.java

```
1 public class ErrorOne {  
2     public static void main(String [] argv) {  
3         System.out.println(" Hello _World" )  
4     }  
5 }
```

4!



# Fehler I

Listing 30: src/no-compile/ErrorOne.java

```
1 public class ErrorOne {  
2     public static void main(String[] argv) {  
3         System.out.println(" Hello World")  
4     }  
5 }
```

```
[mario src]$ javac src/no-compile/ErrorOne.java
```

```
-----  
1. ERROR in src/no-compile/ErrorOne.java (at line 3)  
System.out.println("Hello World")  
                        ^
```

```
Syntax error, insert ";" to complete BlockStatements
```

```
-----  
1 problem (1 error)
```

Listing 31: src/no-compile/ErrorTwo.java

```
1 public class ErrorTwo {  
2     public static void main(String [] argv) {  
3         int solution = 3 + 5;  
4         System.out.println("3+_5_" + soluton);  
5     }  
6 }
```

4!

# Fehler II

Listing 32: src/no-compile/ErrorTwo.java

```
1 public class ErrorTwo {  
2     public static void main(String[] argv) {  
3         int solution = 3 + 5;  
4         System.out.println("3_+_5_=" + soluton);  
5     }  
6 }
```

```
[mario src]$ javac src/no-compile/ErrorTwo.java
```

```
-----  
1. ERROR in src/no-compile/ErrorTwo.java (at line 4)  
System.out.println("3 + 5 =" + soluton);  
                        ^^^^^^^
```

```
soluton cannot be resolved
```

```
-----  
1 problem (1 error)
```

Listing 33: src/no-compile/ErrorThree.java

```
3  int dayOfWeek = 1;
4  switch (dayOfWeek) {
5      case 1:
6          String dayName = "Monday";
7          break;
8          // ...
9  }
10 System.out.println("Weekday _=_ " + dayName);
```

## Fehler III: Compilermeldung

```
[mario src]$ javac src/no-compile/ErrorThree.java
```

```
-----
```

```
1. ERROR in src/no-compile/ErrorThree.java (at line 10)
```

```
System.out.println("Weekday = " + dayName);
```

```
~~~~~
```

```
dayName cannot be resolved
```

```
-----
```

Listing 34: src/no-compile/ErrorThree.java

```
4  switch (dayOfWeek) {  
5      case 1:  
6          String dayName = "Monday";  
7          break;  
8          // ...  
9      }  
10 System.out.println("Weekday _=_ " + dayName);
```

Listing 35: src/no-compile/ErrorThree.java

```
4  switch (dayOfWeek) {  
5      case 1:  
6          String dayName = "Monday";  
7          break;  
8          // ...  
9      }  
10 System.out.println("Weekday _=_ " + dayName);
```

- ▶ Variablen gelten nur innerhalb ihres Blockes { }
- ▶ dayName gilt nur innerhalb von switch
- ▶ Lösung: Deklaration von dayName ausserhalb

Listing 36: src/ErrorThreeSolution.java

```
3  int dayOfWeek = 1;
4  String dayName = "";
5  switch (dayOfWeek) {
6      case 1:
7          dayName = "Monday";
8          break;
9      // ...
10 }
11 System.out.println("Weekday _=_ " + dayName);
```



Listing 37: src/ErrorThreeSolution.java

```
3  int dayOfWeek = 1;
4  String dayName = "";
5  switch (dayOfWeek) {
6      case 1:
7          dayName = "Monday";
8          break;
9      // ...
10 }
11 System.out.println("Weekday _=_ " + dayName);
```

## Ausgabe

```
[mario src]$ java ErrorThreeSolution
Weekday = Monday
```

Listing 38: src/ErrorFour.java

3

```
System.out.println( 1 / 0 );
```

4!

## Listing 39: src/ErrorFour.java

```
3 System.out.println( 1 / 0 );
```

### Kein Compilerfehler

```
[mario src]$ javac ErrorFour.java
```

```
[mario src]$
```

## Listing 40: src/ErrorFour.java

```
3 System.out.println( 1 / 0 );
```

### Kein Compilerfehler

```
[mario src]$ javac ErrorFour.java  
[mario src]$
```

### Aber: Laufzeitfehler

```
[mario src]$ java ErrorFour  
Exception in thread "main" java.lang.  
ArithmeticException: / by zero  
at ErrorFive.main(ErrorFour.java:3)
```

# Inhaltsverzeichnis

- 
- 1 Organisatorisches
  - 2 Kompilieren und Ausführen
  - 3 Hello World - das erste Programm
    - Ausführen
  - 4 Abarbeiten von Befehlen
  - 5 Variablen und einfache Typen
  - 6 Operatoren
  - 7 Fallunterscheidungen
  - 8 Kommentare
  - 9 Probleme und Fehlermeldungen
  - 10 Zusammenfassung**

- ▶ Erzeugen und Kompilieren eines Java-Programmes
- ▶ Einfache Konstrukte
  - ▷ Variablen
  - ▷ Bedingungen
  - ▷ Kommentare
- ▶ Compilerfehler und deren Lösung

A large, light gray circular graphic containing a white number '4' followed by a white exclamation mark '!', positioned on the right side of the slide.

# Was kommt jetzt?

- ▶ Feedback hier vorne abgeben
- ▶ 1. Übung im TEL106/TEL206
- ▶ Danach: Essen
- ▶ Danach: 2. Vorlesung
- ▶ Danach: 2. Übung



4!



A large, bold, black question mark is positioned in the upper left quadrant of the slide. It is set against a background of several light gray, semi-transparent circular and semi-circular shapes that overlap each other.



A large, bold, white number '4' followed by an exclamation mark '!' is centered within a light gray circular shape in the lower right quadrant. The background consists of several light gray, semi-transparent circular and semi-circular shapes that overlap each other.





4!