

# 1 Allgemeine Hinweise

## Inhalt

1.1 Organisatorisches	1-2
1.2 Zur Geschichte von Java	1-8
1.3 Java-Technologien	1-9

# Java - Eine pragmatische Einführung

Holger Arndt

Sommersemester 2024



Autoren/Dozenten: Dr. Benedikt Großer (2000), Prof. Dr. Andreas Frommer (2001, 2002, 2007), Dr. Holger Arndt (2003, 2006, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2024), Dr. Werner Hofschuster<sup>†</sup> (2004, 2005), Dr. Martin Galgon (2021, 2022, 2023)

Version: Mittwoch, 3. April 2024, 12:32

## 1.1 Organisatorisches

### Hörerkreis

- **Bachelor Informatik**  
Wahlpflichtveranstaltung im Pflichtmodul INF3: Objektorientierte Programmierung
- **Bachelor Mathematik, PO 2020** mit Nebenfach Informatik  
Wahlpflichtveranstaltung im Pflichtmodul INF3: Objektorientierte Programmierung
- **Bachelor Mathematik, PO bis 2013** mit Nebenfach Informatik  
Wahlpflichtveranstaltung im Pflichtmodul NInf.OoP: Objektorientierte Programmierung
- **Bachelor Wirtschaftsmathematik, PO 2020**  
Wahlpflichtveranstaltung im Wahlpflichtmodul INF3: Objektorientierte Programmierung
- **Bachelor Angewandte Naturwissenschaften, PO 2022** mit Schwerpunktfach Informatik  
Wahlpflichtveranstaltung im Pflichtmodul INF3: Objektorientierte Programmierung
- **Bachelor Angewandte Naturwissenschaften, PO 2012** mit Schwerpunktfach Informatik  
Wahlpflichtveranstaltung im Pflichtmodul Objektorientierte Programmierung
- **Bachelor Angewandte Naturwissenschaften, PO 2007** mit Schwerpunktfach Informatik  
Wahlpflichtveranstaltung im Wahlpflichtmodul I9b: Programmieretechniken/Softwaretechnologie  
Wahlpflichtveranstaltung im Wahlpflichtmodul I9d: Internet-/Webtechnologie

- **Bachelor Informationstechnologie und Medientechnologie**  
Pflichtveranstaltung im Wahlpflichtmodul ObjPr: Objektorientierte Programmierung mit Java
- **Bachelor Informationstechnologie**  
Pflichtveranstaltung im Wahlpflichtmodul Objektorientierte Programmierung mit Java
- **Bachelor Elektrotechnik, PO 2021**  
Wahlpflichtveranstaltung im Wahlpflichtmodul INF3: Objektorientierte Programmierung
- **Bachelor Elektrotechnik, PO 2016**  
Wahlpflichtveranstaltung im Wahlpflichtmodul OoP: Objektorientierte Programmierung
- **Bachelor Wirtschaftsingenieurwesen Elektrotechnik, PO 2021**  
Wahlpflichtveranstaltung im Wahlpflichtmodul INF3: Objektorientierte Programmierung
- **Bachelor Wirtschaftsingenieurwesen Elektrotechnik, PO bis 2017**  
Wahlpflichtveranstaltung im Wahlpflichtmodul OoP: Objektorientierte Programmierung
- **Master Wirtschaftsingenieurwesen Informationstechnik, PO bis 2017**  
Wahlpflichtveranstaltung im Wahlpflichtmodul OoP: Objektorientierte Programmierung

- **Kombinatorischer Bachelor** mit Teilstudiengang Informatik  
Wahlpflichtveranstaltung im Pflichtmodul INF3: Objektorientierte Programmierung
- **Kombinatorischer Bachelor** mit Teilstudiengang Elektrotechnik  
Wahlpflichtveranstaltung im Wahlpflichtmodul INF3: Objektorientierte Programmierung
- **Kombinatorischer Bachelor, Optionalbereich, PO 2021**  
Wahlpflichtkomponente im Wahlpflichtmodul INF301: Informatik 1  
und Wahlpflichtkomponente im Wahlpflichtmodul INF302: Informatik 2
- **Bachelor Physik, PO 2019**  
Wahlpflichtveranstaltung im Wahlpflichtmodul INF3: Objektorientierte Programmierung
- **Bachelor Physik, PO bis 2013**  
Wahlpflichtveranstaltung im Wahlpflichtmodul OoP: Objektorientierte Programmierung im Pflichtmodul BW2 (Bachelor-Wahlpflichtfach 2)

### Termine

Vorlesung:	Di., 14:15–15:45	Hörsaal 11
Übung (Gruppe 1):	Mo., 10:15–11:45	G.16.15
Übung (Gruppe 2):	Mo., 14:15–15:45	G.16.15
Übung (Gruppe 3):	Di., 10:15–11:45	G.16.15
erste Übung:	Mo., 15.04. bzw. Di., 16.04.	

### Anmeldung zur Übung und Einteilung in Gruppen

Anmeldung: in StudiLöwe mit Angabe von Prioritäten für die Gruppen  
vom 08.04.2024 bis zum 11.4.2024, 12:00 Uhr

Einteilung: im Anschluss, rechtzeitig vor der ersten Übung

### Übungsblätter

Ausgabe: zweiwöchentlich in Moodle (insgesamt sechs Blätter)

Bearbeitung: Zweiergruppen erlaubt

Abgabe: über Moodle (Details auf Übungsblatt 1)

### Klausur

- Erwerb der Leistungspunkte durch eine 90-minütige Klausur
- Die Klausur ist geplant für Dienstag, 17.09.2024, 11:30–13:00 Uhr. Der endgültige Termin wird in Moodle verkündet werden.
- Die Nachklausur findet nach dem Wintersemester 2024/2025 statt.
- Zum Bestehen werden 38 von 75 möglichen Punkten benötigt.
- Bonuspunkte:

Punkte aus Übungen	Notenverbesserung
≥ 50 %	–0,3 oder –0,4
≥ 65 %	–0,6 oder –0,7
≥ 80 %	–1,0

- Der Bonus wird nur angerechnet, wenn die Klausur selbst bestanden ist, d. h. mindestens Note 4,0.
- Der Bonus gilt für die Hauptklausur und die Nachklausur, die direkt auf das Semester des Punkteerwerbs folgen. Danach verfallen die Bonuspunkte.





### Moodle-Kurs zur Vorlesung

<https://moodle.uni-wuppertal.de/course/view.php?id=35056>

In Moodle finden Sie:

- Schwarzes Brett mit Ankündigungen
- Folien und komplette Quelltexte
- Übungsblätter

## 1.2 Zur Geschichte von Java

1992	Sun Microsystems entwickelt „*7“ (PDA) mit Green-OS und Interpreter Oak.	
Apr. 1993	NCSA Mosaic (erster grafischer Web-Browser)	
März 1995	Hotjava (Browser von Sun, der Java-Applets ausführen kann)	
23. Mai 1995	offizieller Geburtstag von Java	
Dez. 1995	Netscape 2.0 (mit eingebautem Java-Interpreter)	
23. Jan. 1996	JDK 1.0	
18. Feb. 1997	JDK 1.1 (JDBC, Beans, ...)	
Anfang 1998	Browserunterstützung für Java 1.1	
8. Dez. 1998	JDK 1.2, seither Java 2 Platform genannt (Just-In-Time-Compiler, Swing, Java2D, ...)	
8. Mai 2000	JDK 1.3 (Performanceverbesserungen, ...)	
6. Feb. 2002	JDK 1.4 (Assertions, ...)	
30. Sept. 2004	JDK 5 (ursprünglich als JDK 1.5.0; Generics, erweiterte for-Schleife, Autoboxing, Enumerations, neue Klassen für Synchronisation von Threads, Annotationen, ...)	
11. Dez. 2006	JDK 6 (Zusammenarbeit mit Skriptsprachen, Performanceverbesserungen, ...)	
2006/2007	weitgehende Freigabe von Java als Open Source im Rahmen des OpenJDK-Projekts	
27. Jan. 2010	Übernahme von Sun Microsystems durch Oracle	
28. Juli 2011	JDK 7 (neues Dateisystem-API, Autocloseables, Typinferenz, Multi-Catch, switch für Strings)	
18. März 2014	JDK 8 (Lambda-Ausdrücke, ...)	

21. Sept. 2017	JDK 9 (Modularisierung, ...)
20. März 2018	JDK 10 (Typinferenz für lokale Variable, ...)
25. Sept. 2018	JDK 11 (Ausführung von „Java-Skripten“ ohne Compilierung, Wegfall einiger Packages aus dem Enterprise-Modul, ...)
19. März 2019	JDK 12
17. Sept. 2019	JDK 13
17. März 2020	JDK 14 (Switch Expressions, ...)
15. Sept. 2020	JDK 15 (Textblöcke)
16. März 2021	JDK 16 (Records)
14. Sept. 2021	JDK 17 (Sealed Classes)
22. März 2022	JDK 18 (Simple Web Server)
20. Sept. 2022	JDK 19
21. März 2023	JDK 20
19. Sept. 2023	JDK 21 (Pattern Matching für switch, Virtual Threads)
19. März 2024?	JDK 22 (Unterstrich als unbenannte Variable, Foreign Function and Memory API)

## 1.3 Java-Technologien

Java SE: Java Platform, Standard Edition (Core/Desktop)

Java EE: Java Platform, Enterprise Edition (Enterprise/Server), seit 2019 als Jakarta EE

Java ME: Java 2 Platform, Micro Edition (Mobile/Wireless)

Java Card (Chipkarten)

## Literatur

- [Blo18] Joshua Bloch. **Effective Java**. 3. Aufl. Programmieren mit Java. Heidelberg: dpunkt.verlag, Sep. 2018. ISBN: 978-3-86490-578-0. URL: <https://content-select.com/de/portall/media/view/5d76655a-3760-4d43-ab3a-7e37b0dd2d03>.
- [Eck22] David J. Eck. **Introduction to Programming Using Java**. 9. Swing Edition. Lulu, Mai 2022. URL: <https://math.hws.edu/javanotes-swing/>.
- [KH14] Guido Krüger und Heiko Hansen. **Java-Programmierung - Das Handbuch zu Java 8**. 8. Aufl. Köln: O'Reilly Verlag, Mai 2014. ISBN: 978-3-95561-514-7. TYD 441, 7. Aufl. (2011) online: URL: <http://www.javabuch.de/download.html>.
- [Kof22] Michael Kofler. **Java: Der Grundkurs**. 4. Aufl. Bonn: Rheinwerk Computing, Mai 2022. ISBN: 978-3-8362-8392-2. (3. Aufl.: TYD 632). URL: <https://ebookcentral.proquest.com/lib/ubwuppertal-ebooks/detail.action?docID=6970458>.
- [Lia99] Sheng Liang. **The Java Native Interface: Programmer's Guide and Specification**. 1. Aufl. Reading: Addison Wesley, Juni 1999. ISBN: 0-201-32577-2. URL: <https://web.archive.org/web/20120106014712/http://java.sun.com:80/docs/books/jni/>.
- [Rog01] Axel Rogat. **Objektorientiertes Programmieren mit C++ und Java**. 2.1. Wuppertal: Hochschulrechenzentrum Universität Wuppertal, 2001.

- [Sav17] Walter J. Savitch. **Java: An Introduction to Problem Solving and Programming**. 8. Aufl. Pearson, Juni 2017. ISBN: 978-0134710754.
- [Sch10] Reinhard Schiedermeier. **Programmieren mit Java**. 2. Aufl. München: Pearson Studium, 2010. ISBN: 978-3-86894-031-2.
- [Sch13] Reinhard Schiedermeier. **Programmieren mit Java II**. München: Pearson, 2013. ISBN: 978-3-8689-4129-6. TYD 575. URL: <http://sol.cs.hm.edu/4129/>.
- [SM16] Walter J. Savitch und Kenrick Mock. **Absolute Java**. 6. Aufl. Boston: Pearson Education Limited, 2016. ISBN: 978-1292109220.
- [Ull18] Christian Ullenboom. **Java SE 9 Standard-Bibliothek**. 3. Aufl. Bonn: Rheinwerk Computing, 2018. ISBN: 978-3-8362-5874-6. TYD 581, 2. Aufl. („Java SE 8 Standard-Bibliothek“, 2014) online: URL: <https://openbook.rheinwerk-verlag.de/java8/>.
- [Ull24] Christian Ullenboom. **Java ist auch eine Insel**. 17. Aufl. Bonn: Rheinwerk Computing, 2024. ISBN: 978-3-8362-9544-4. 16. Aufl. (2022): <https://ebookcentral.proquest.com/lib/ubwuppertal-ebooks/detail.action?docID=6807940>, 15. Aufl. (2020) online: <https://openbook.rheinwerk-verlag.de/javainsel/>. 16. Aufl.: TYD 635.

## Links

- Download aktueller JDK-Versionen:  
<https://jdk.java.net/>
- API-Referenz zu aktuellen JDK-Versionen:  
<https://docs.oracle.com/en/java/javase/>
- The Java Tutorials:  
<https://docs.oracle.com/javase/tutorial/>

## 2 Erste Schritte

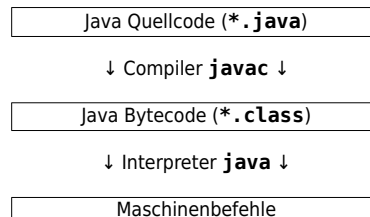
### Inhalt

2.1	Die virtuelle Maschine	2-2
2.2	„Hello World“ als Applikation	2-3
2.3	Ein grafisches „Hello World“ mit Swing	2-7
2.4	JShell	2-8
2.5	Java-Skripte	2-9
2.6	Zusammenfassende Fragen	2-10

**Inhalt/Ziele:** Ziel dieses Abschnittes ist das Erlernen der grundlegenden Begriffe und Arbeitsschritte der Programmiersprache Java. Dabei werden die Werkzeuge des Java-Development-Kits **javac** und **java** vorgestellt.

## 2.1 Die virtuelle Maschine

Java-Programme werden nicht direkt in Maschinencode übersetzt, sondern in einen **plattformunabhängigen Bytecode**. Dieser wird zur Laufzeit von einer **Java Virtual Machine** interpretiert oder durch einen **Just-in-Time**-Compiler weiter übersetzt.



## 2.2 „Hello World“ als Applikation

- Quelltext: **Hello1.java**
- Compilieren: **javac Hello1.java**
- dabei entsteht: **Hello1.class**
- Ausführen: **java Hello1**

HelloWorld/Hello1.java

```
1 /**  
2  Hello-World-Applikation  
3  @author Holger Arndt  
4  @version 13.02.2003  
5  */  
6 public class Hello1  
7 {  
8     public static void main(String[] args)  
9     {  
10         System.out.println("Hello_World!");  
11     }  
12 }
```

- Basis jeder Java-Applikation
- Klassen bilden die grundlegenden Bausteine von Java. Eine Klassendefinition erfolgt mit dem Schlüsselwort **class**.
- Das Schlüsselwort **public** bezieht sich auf Zugriffsrechte.
- Der Compiler **javac** erwartet, dass die Java-Quelltextdatei mindestens eine **class**-Definition gleichen Namens beinhaltet. (Also: Eine Java-Quelltextdatei **Xyz.java** enthält mindestens die Klasse **class Xyz**.) Es ist erlaubt, aber nicht üblich, mehrere Klassen in einer Java-Quelltextdatei zu definieren.
- Eine Java-Applikation muss eine **class**-Definition enthalten, die die Methode (Funktion) **main()** implementiert.
- Der Interpreter **java** benutzt die Methode **main()** als Startpunkt für den Programmablauf. Er arbeitet alle Anweisungen der **main()**-Methode ab.
- Die Applikation endet nach Ausführung der letzten Anweisung der **main()**-Methode.

### Quelltextanalyse

- Definition einer Klasse:

```
6 | public class Hello1
7 | {
12 | }
```

- Mit dem Schlüsselwort **class** beginnt die Definition einer Klasse, in diesem Falle **Hello1**.
- Ein Paar geschweifeter Klammern begrenzt den Klassenkörper, in dem wir dann Klassenattribute und -methoden implementieren können. Es folgt kein Semikolon.

- die Methode **main()**:

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

- Jede Java-Applikation muss eine **main()**-Methode enthalten, die wie oben angegeben definiert wird. Der Interpreter **java** sucht für den Programmeinstieg genau nach dieser Signatur.
- Dem Methodennamen **main()** sind drei Schlüsselwörter vorangestellt:
  - \* **public** bezieht sich auf **Zugriffsrechte**.
  - \* **static** zeigt an, dass **main()** eine **Klassenmethode** ist.
  - \* **void** bezieht sich auf den **Datentyp/Rückgabewert**.
- Hinter dem Methodennamen **main()** schließt sich in runden Klammern noch eine Argumentliste an: **String[] args**. Mit diesem Mechanismus kann das Betriebssystem beim Starten der Applikation Kommandozeilenargumente übergeben.

- Ausgabe von Text:

```
10 | System.out.println("Hello_World!");
```

### Java-Versionen auf den IT-Rechnern

Version	Verzeichnis der Binaries
Java 22 (OpenJDK)	<b>/usr/local/sw/jdk-22/bin</b>
Java 21 (Oracle)	<b>/usr/local/sw/jdk-21/bin</b>
Java 17 (OpenJDK)	im Pfad voreingestellt
Java 14 (OpenJDK)	<b>/usr/local/sw/jdk-14/bin</b>
Java 11 (OpenJDK)	<b>/usr/lib64/jvm/java-11-openjdk/bin</b>
Java 8 (OpenJDK)	<b>/usr/lib64/jvm/java-1.8.0-openjdk/bin</b>

Um immer mit Version 22 zu arbeiten, können Sie der Datei **.bashrc** in Ihrem Home-Verzeichnis folgende Zeile hinzufügen:

```
export PATH=/usr/local/sw/jdk-22/bin:$PATH
```

## 2.3 Ein grafisches „Hello World“ mit Swing

### HelloWorld/HelloSwing.java

```
1 /**
2  * Grafische Hello-World-Applikation mit Swing
3  * @author Holger Arndt
4  * @version 15.03.2018
5  */
6
7 import javax.swing.*;
8
9 public class HelloSwing extends JFrame
10 {
11     private HelloSwing()
12     {
13         add(new JLabel("<html><big>Hello_World!</big></html>",
14                        SwingConstants.CENTER));
15         setDefaultCloseOperation(EXIT_ON_CLOSE);
16         setTitle("Halloooo");
17         setSize(200, 100);
18         setVisible(true);
19     }
20
21     public static void main(String[] args)
22     {
23         SwingUtilities.invokeLater(HelloSwing::new);
24     }
25 }
```

2.4 JShell

Seit Java 9 lassen sich Java-Kommandos interaktiv in der **JShell** testen.

```
wmai20 ~ > jshell
| Willkommen bei JShell - Version 22
| Geben Sie für eine Einführung Folgendes ein: /help intro

jshell> 2 + 3 * (4 + 5)
$1 ==> 29

jshell> Math.sqrt(2)
$2 ==> 1.4142135623730951

jshell> int summe = 0
summe ==> 0

jshell> summe += 42
$4 ==> 42

jshell> new Date()
$5 ==> Wed Mar 20 11:42:01 CET 2024

jshell> for (int i = 1; i <= 10; i++)
...> System.out.print(" " + i * i)
1 4 9 16 25 36 49 64 81 100
jshell> /exit
| Auf Wiedersehen
```

2.5 Java-Skripte

Seit Java 11 können Programme, die nur in einer Quelltextdatei stehen, ohne Compilierung ausgeführt werden:

```
wmai20 ~/Java/Vorlesung/Quellen/HelloWorld > rm Hello1.class
wmai20 ~/Java/Vorlesung/Quellen/HelloWorld > java Hello1.java
Hello World!
```

Seit Java 22 klappt das auch für Programme, die auf mehrere Quelltextdateien verteilt sind.

Unter Linux lässt sich auch eine Shebang-Zeile einfügen und der Quelltext ausführbar machen. Die Endung darf dabei nicht **.java** sein.

```

HelloWorld/Hello1a.javash
1  #!/usr/bin/java --source 11
7  public class Hello1a
8  {
9      public static void main(String[] args)
10     {
11         System.out.println("Hello_World!");
12     }
13 }

wmai20 ~/Java/Vorlesung/Quellen/HelloWorld > ./Hello1a.javash
Hello World!
```

2.6 Zusammenfassende Fragen

- Warum bezeichnet man Java als plattformunabhängige Sprache?
- Wie implementiert und startet man eine „Hello World!“-Applikation?
- Wie kann man in einer Applikation Text auf den Bildschirm drucken lassen?

3 Grundlagen kurzgefasst

**Inhalt**

<b>3.1</b>	<b>Ein- und Ausgabe</b>	<b>3-2</b>
<b>3.2</b>	<b>Lexikalische Struktur</b>	<b>3-4</b>
<b>3.3</b>	<b>Einfache Datentypen und Variablen</b>	<b>3-7</b>
<b>3.4</b>	<b>Referenzdatentypen</b>	<b>3-11</b>
3.4.1	Arrays	3-12
3.4.2	Strings	3-14
3.4.2.1	Textblöcke	3-15
3.4.3	Identität und Äquivalenz	3-16
3.4.4	Aufzählungen mit enum	3-17
<b>3.5</b>	<b>Typkonvertierungen</b>	<b>3-18</b>
<b>3.6</b>	<b>Operatoren und Ausdrücke</b>	<b>3-19</b>
<b>3.7</b>	<b>Mathematische Funktionen, statische Importe</b>	<b>3-23</b>
<b>3.8</b>	<b>Kontrollstrukturen</b>	<b>3-24</b>
<b>3.9</b>	<b>Kommandozeilenparameter</b>	<b>3-32</b>
<b>3.10</b>	<b>Zusammenfassende Fragen</b>	<b>3-33</b>

**Inhalt/Ziele:** Ziel dieses Abschnittes ist die Beschreibung der elementaren Bestandteile eines Java-Programms. Im Anschluss an diesen Abschnitt sollten wir einfache Javaprogramme schreiben und kompliziertere zumindest lesen können.

## 3.1 Ein- und Ausgabe

Java ermöglicht den Zugriff auf die Standardein- und ausgabe über **System.in** und **System.out**.

### Grundlagen/EinAusgabe/SystemOut.java

```

6 public class SystemOut
7 {
8     public static void main (String[] argv)
9     {
10         System.out.println("6*7=" + (6 * 7));
11     }
12 }
```

Das Einlesen von der Standardeingabe ist etwas mühsamer. Wir werden uns später ausführlich damit beschäftigen (siehe 11.2). Das folgende Programm liefert nicht das erwünschte Ergebnis!

### Grundlagen/EinAusgabe/SystemIn.java

```

14 int val = System.in.read();
15 byte b = (byte) val;
16 char c = (char) val;
17 System.out.println("Eingabe als int: " + val);
18 System.out.println("Eingabe als byte: " + b);
19 System.out.println("Eingabe als char: " + c);
```

Einfache Datentypen können am leichtesten mit Hilfe der Klasse **Scanner** eingelesen werden. Zusätzlich ist eine formatierte Ausgabe mit **printf** im Stil der Programmiersprache C möglich.

### Grundlagen/EinAusgabe/IOJava5.java

```

7 import java.util.Scanner;
13 double d;
14 int i;
15 String s;
16
17 Scanner sc = new Scanner(System.in);
18 d = sc.nextDouble();
19 i = sc.nextInt();
20 s = sc.next();
21
22 System.out.printf("d=%f,i=%d,s=%s\n", d, i, s);
```

## 3.2 Lexikalische Struktur

- kein Präprozessor
- Eingabezeichen aus dem Unicode-Zeichensatz

### HelloWorld/HelloUnicodeSwing.java

```

53 String Chinese_中文_普通话_汉语 = "你好";
61 String Czech_čestina = "Dobrý den";
75 String Esperanto = "Saluton (Ĝoŝanĝo ĉiujaŭde)";
85 String German_Deutsch = "Guten Tag / Grüß Gott";
91 String Greek_ελληνικά = "Γειά σας";
105 String Hindi_हिंदी = "नमस्ते / नमस्कार";
137 String Mathematics = "VpEworld*hellop";
189 String Thai_ภาษาไทย = "สวัสดีครับ / สวัสดีค่ะ";
199 String Ukrainian_українська = "Вітаю / Добрий день! / Привіт";
```

- Auf Rechnern, die noch keine Unicode-Umgebung haben, muss obiges Programm mit der Option **-encoding utf-8** übersetzt werden.
- Umgekehrt müssen auf Unicode-Systemen alte Latin-1-Programme mit **-encoding iso-8859-1** übersetzt werden.
- Alle Beispielprogramme der Vorlesung sind in UTF-8 codiert.

- Die folgenden Wörter sind in Java reserviert (Schlüsselwörter, keywords):

**abstract, assert, boolean, break, byte, case, catch, char, class, continue, default, do, double, else, enum, extends, final, finally, float, for, if, implements, import, instanceof, int, interface, long, native, new, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, try, void, volatile, while**

reservierte Wörter, die in der aktuellen Version nicht verwendet werden:

**const, goto, strictfp**

keine Schlüsselwörter, sondern reservierte Literale:

**false, null, true,**

kontextabhängige Schlüsselwörter, die nur in bestimmten Situationen reserviert sind:

**exports, module, non-sealed, open, opens, permits, provides, record, requires, sealed, to, transitive, uses, var, when, with, yield**

- Namenskonventionen:
  - Klassen: **Hello, Button, Color, ArrayIndexOutOfBoundsException**
  - Methoden: **main, setLabel, toString, printStackTrace**
  - Attribute: **argv, height, startAngle**
  - „Konstanten“: **PI, MIN\_VALUE, MAX\_PRIORITY**
- Kommentare:
  - einzeilige Kommentare: **// xxx**
  - mehrzeilige Kommentare: **/\* xxx \*/**
  - Dokumentationskommentare: **/\*\* xxx \*/**

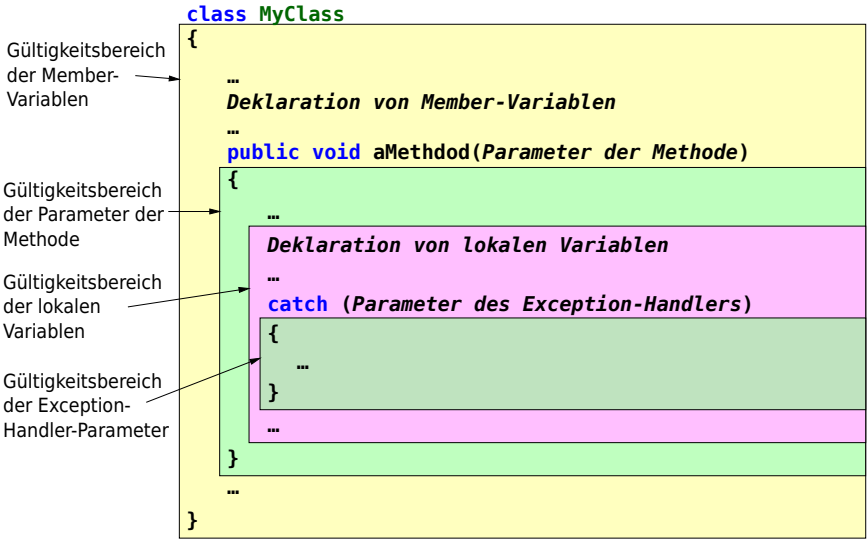
Die ein- und mehrzeiligen Kommentare sind sinnvoll für die **Programmierer** von Klassen. Die Dokumentationskommentare sind für **Benutzer** von Klassen gedacht und arbeiten mit **javadoc** (siehe 4.1.1) zusammen.

3.3 Einfache Datentypen und Variablen

Die acht einfachen **Datentypen** in Java zeichnen sich durch feste Länge und Standardwerte aus.

Typname	Länge in Byte	Wertebereich	Standardwert
boolean	1	true, false	false
char	2	alle Unicode-Zeichen	'\000' bzw. '\u0000'
byte	1	-2 <sup>7</sup> , ..., 2 <sup>7</sup> - 1	0
short	2	-2 <sup>15</sup> , ..., 2 <sup>15</sup> - 1	0
int	4	-2 <sup>31</sup> , ..., 2 <sup>31</sup> - 1	0
long	8	-2 <sup>63</sup> , ..., 2 <sup>63</sup> - 1	0
float	4	±1,4024 · 10 <sup>-45</sup> , ..., ±3,4028 · 10 <sup>38</sup>	0.0f
double	8	±4,0407 · 10 <sup>-324</sup> , ..., ±1,7977 · 10 <sup>308</sup>	0.0

- Java kennt keine expliziten Zeiger oder Bitfelder.
- Eine **Variable** ist gekennzeichnet durch:
- Datentyp:** Neben einfachen Datentypen lernen wir später noch Referenzdatentypen kennen.
- Name:** siehe Namenskonventionen
- Scope** (deutsch: Gültigkeitsbereich, Lebensraum): Man unterscheidet:
- Member-Variablen
  - lokale Variablen
  - Variablen aus Parameterlisten
  - Exception-Handler-Variablen





**Wert:** Der Compiler **javac** lässt nicht zu, dass eine nicht mit einem Wert initialisierte lokale Variable einer anderen zugewiesen wird. Man kann das Anlegen und Initialisieren zusammenfassen. Wird eine Member-Variable deklariert, so erhält sie den Standardwert des Datentyps.

#### Grundlagen/Datentypen/TypBsp.java

```

10  int a;
11  a = 3;
12  int b = 5;
13  int c = a * b;
14  int d;
15  //c = a * d; // Fehler: d ist nicht initialisiert

```

Mit dem Schlüsselwort **final** kann man unterbinden, dass eine Variable nach der ersten Belegung nochmals geändert wird.

```

16  final int e = 4;
17  //e = 5; // Fehler: e ist final

```

Mit Java 10 wurde **Typinferenz** für lokale Variable eingeführt. Werden lokale Variablen direkt initialisiert, kann der Compiler den Typ daraus ableiten und statt des Typs kann das reservierte Wort **var** verwendet werden.

```

19  var zahl = 25; // → int
20  var wert = 3.14; // → double

```

## 3.4 Referenzdatentypen

Neben den einfachen Datentypen gibt es in Java noch die Referenzdatentypen. Darunter fallen alle **Objekte**, aber auch **Arrays** und **Strings**.

- Diese Typen werden „per Referenz“ verarbeitet, d. h. die Adresse des Objekts (Arrays oder Strings) wird in einer Variablen abgelegt.
- Referenzdatentypen werden zur Laufzeit erzeugt.
- Variablen vom Referenzdatentyp haben den Standardwert **null**.
- Objekte können ihrerseits wieder über Methoden und Variablen verfügen.

### 3.4.1 Arrays

Die Deklaration einer Array-Variablen erfolgt, indem man an einen einfachen Datentyp ein Paar eckiger Klammern anhängt:

```

22  double[] daten;

```

Bis jetzt ist noch nicht spezifiziert, wie viele Einträge das Array haben soll. Mit dem **new**-Operator können wir das festlegen und entsprechenden Speicherplatz anfordern:

```

23  daten = new double[10];

```

Man kann Deklaration und die so genannte **Instantiierung** wieder zusammenfassen:

```

25  int[] zensuren = new int[6];

```

Referenzierung von Array-Elementen erfolgt mit Hilfe der eckigen Klammern. Dabei wird das erste Element mit **0** angesprochen.

```

26  zensuren[0] = 2;
27  zensuren[1] = 4;

```

Man kann Deklaration, Instantiierung und Initialisierung auch zusammenfassen, ohne den **new**-Operator zu benutzen:

```

29  int[] zensuren2 = { 2, 4, 5, 3, 2, 1 };

```

Als Referenzdatentypen verfügen Arrays über das Attribut **length**, das die Länge des Arrays zum Inhalt hat.

```

30  System.out.println(zensuren2[zensuren2.length - 1]); // druckt 1

```

Mehrdimensionale Arrays sind möglich:

```

32  char[][] schachbrett = new char[8][8];

```

Die Klammern **[]** können auch hinter der Variable angegeben werden.

```

35  double daten2[] = new double[12];

```

Mit der Methode **arraycopy** aus **System** lassen sich (Teile eines) Arrays kopieren.

```

36  System.arraycopy(daten, 2, daten2, 5, 4);
37  // daten2[5..8] = daten[2..5]

```

### 3.4.2 Strings

Strings (Zeichenketten) werden in Java ebenfalls als Referenzdatentypen verwendet.

Deklaration und Initialisierung separat:

```
39 String dateiname1;
40 dateiname1 = "Hello1.java";
```

Deklaration und Initialisierung zusammengefasst:

```
42 String dateiname2 = "Hello1.java";
```

Der Referenzdatentyp **String** verfügt über die einzigen überladenen Operatoren (+, +=) in Java:

```
44 String rakete = "Apollo";
45 rakete = rakete + 3 * 4;
46 System.out.println(rakete); // druckt Apollo12
```

Als Referenzdatentypen verfügen Strings über Methoden wie **length()** und **charAt()**.

```
47 char x = rakete.charAt(1);
48 System.out.println("Zeichen_1_=" + x); // druckt Zeichen 1 = p
49 System.out.println("Zeichenanzahl_=" + rakete.length()); // druckt ... = 8
```

Prinzipiell ändern Strings ihren Wert **nie**. In Java sind dennoch Zuweisungen auf Strings erlaubt. Intern legt der Compiler dann jeweils einen neuen String an.

Siehe API-Referenz, um mehr über Strings zu erfahren.

#### 3.4.2.1 Textblöcke

Seit Java 15 können mehrzeilige **Textblöcke** in dreifache Anführungszeichen eingeschlossen werden. Enthaltene Zeilenumbrüche und einfache Anführungszeichen brauchen dann keine Sonderbehandlung mehr. Die beiden Strings im folgenden Beispiel sind gleich:

##### Grundlagen/Datentypen/TextBlockBsp.java

```
10 String s1 = "public static void main(String[] args)\n"
11 + "    {\n"
12 + "        System.out.println(\"Hello World!\");\n"
13 + "    }\n";
14 String s2 = ""
15     public static void main(String[] args)
16     {
17         System.out.println("Hello World!");
18     },
19 ;
20 System.out.println(s1);
21 System.out.println(s2);
```

### 3.4.3 Identität und Äquivalenz

Bei Referenzdatentypen muss man zwischen **Identität** und **Äquivalenz** unterscheiden:

- Identität prüft, ob zwei Referenzen auf die gleiche Adresse verweisen. Der zugehörige Operator lautet **==**.
- Äquivalenz prüft, ob zwei Adressen gleichen Inhalt haben. Die zugehörige Methode lautet **equals()**.

##### Grundlagen/IdentitätÄquivalenz/IÄcheck.java

```
28 String wort1 = "Color.java";
29 String wort2 = "Color";
30 boolean icheck, ächeck;
31 wort2 = wort2 + ".java";
32
33 icheck = (wort1 == wort2); // false (zwei verschiedene Objekte)
34 ächeck = (wort1.equals(wort2)); // true (aber mit gleichem Inhalt)
35 wort1 = wort2; // (jetzt verweisen wort1, wort2 auf das gleiche Objekt)
36
37 icheck = (wort1 == wort2); // true
38 wort1 = new String(wort2);
39 icheck = (wort1 == wort2); // false
```

### 3.4.4 Aufzählungen mit enum

Das reservierte Schlüsselwort **enum** wird für typsichere Aufzählungen benutzt (als **enum**-Klasse).

Beispiel: Erzeuge Konstanten für die Wochentage.

##### Grundlagen/Datentypen/EnumBsp.java

```
7 enum Wochentage
8 {
9     montag, dienstag, mittwoch, donnerstag, freitag, samstag, sonntag;
10 }
11
12 Wochentage tage = Wochentage.samstag;
13
14 switch (tage)
15 {
16     case samstag:
17     case sonntag: System.out.println("Wochenende. Party.");
18 }
```