

ISW: Software Engineering WS 2023/24

Java: Grundlagen und Beispiele

Michael Anders

Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 205
69120 Heidelberg, Germany

<https://se.ifi.uni-heidelberg.de>



■ Teil 1: Überblick

- Erstkontakt
- Java Standard Edition
- Objektorientierte Programmierung (OOP)
- Unterschiede zwischen C++ und Java

■ Teil 2: Java Basics

- Klassen
- Felder (Arrays)
- Methoden
- Konstruktoren
- Vererbung
- Objekte
- Ausnahmen
- Collections
- Callbacks, Lambda, Methoden-Referenzen, ...

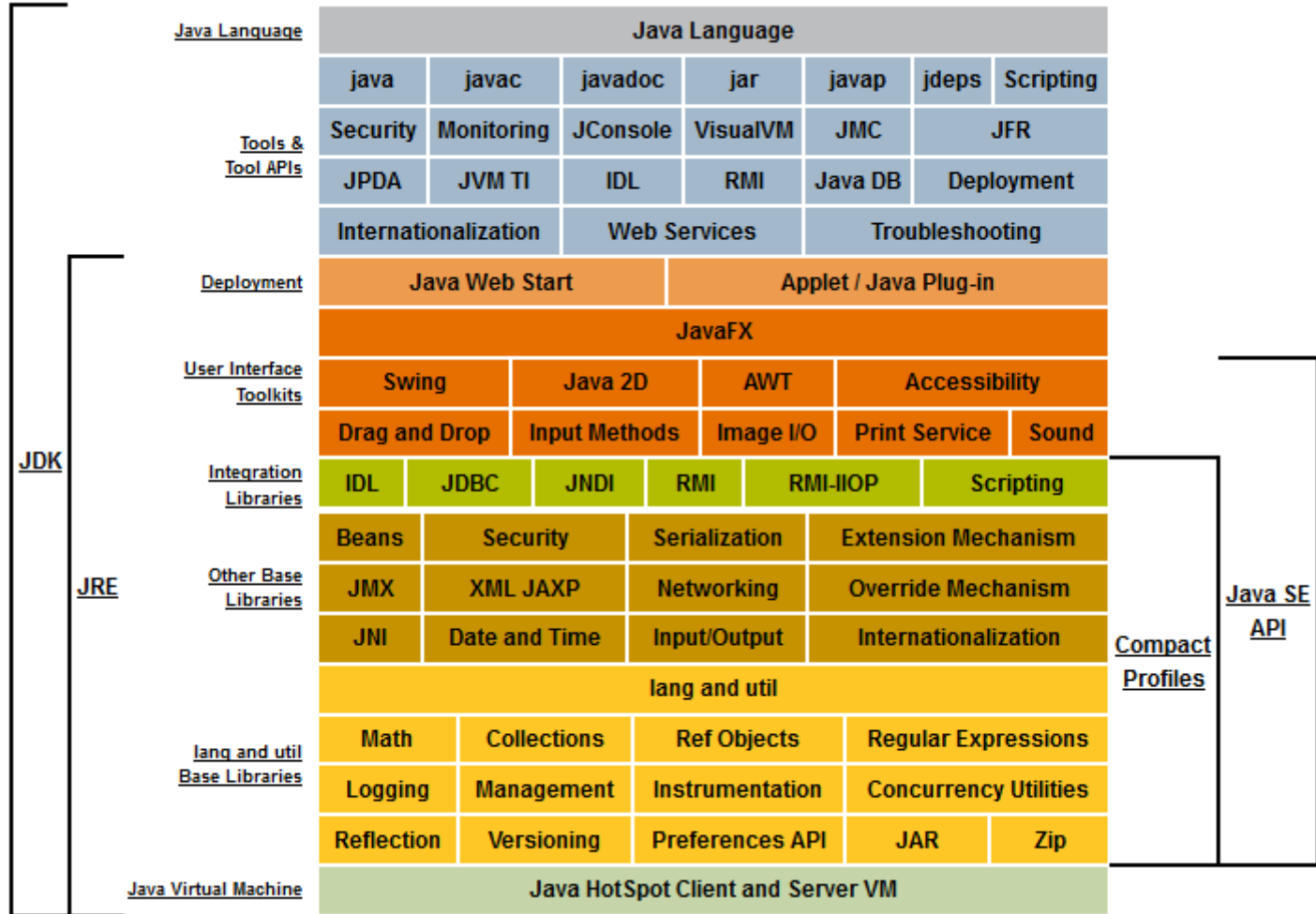
- Was ist Java(tm)?
 - Eine Plattform
 - Eine Programmiersprache

- Wer ist der Hersteller?
 - Ursprünglich Sun Microsystems
 - Inzwischen von Oracle aufgekauft

- Website?
 - <https://www.java.com/>
 - <https://www.oracle.com/technetwork/java/index.html>
 - ...

- Was bedeutet Plattform?
 - Eine hardware-spezifische Umgebung in der Anwendungen ausgeführt werden
- Welche Java Plattformen gibt es?
 - Enterprise Edition (JEE)
 - Anwendungen für Server
 - Standard Edition (JSE)
 - Anwendungen für Desktops
 - Micro Edition (JME)
 - Anwendungen für mobile und eingebettete Geräte im Internet of Things (IoT)
 - Java Card
 - Anwendungen für Smartcards
- Was kennzeichnet die Plattformen?
 - Plattform-spezifische Klassenbibliothek
 - Plattform-spezifischer Interpreter (Java Virtual Machine)
- Allerdings: plattform-übergreifende Programmiersprache und Compiler

Components and Technologies of Java



<https://www.oracle.com/java/technologies/platform-glance.html>

- Was kann man mit den Plattformen entwickeln?
 - **Applications:**
 - Standalone
 - Mit Graphical User Interface
 - Ohne Graphical User Interface
 - Server: Datenbank-Server, Application-Server, ...
 - Clients, die sich zu einem Server verbinden uvm.
 - **Applets:** Anwendungen, die in eine Webseite eingebettet sind
 - **Servlets:** Anwendungen, die in einen Web Server eingebettet sind und HTTP requests empfangen, bearbeiten und zurückgeben
 - **JavaServer Pages und JavaServer Faces:** Webseiten, die Java Quellcode enthalten und bei Anfrage der Seite kompiliert werden
 - **Enterprise Java Beans:** Komponenten für J2EE Application Server
 - **Web Services:** z.B. REST-Schnittstelle
 - **MIDlets:** Programme für Mobile Information Device Profile
 - Klassenbibliotheken
 - **Portlets und Remote Portlets:** Komponenten für Portal-Server

- Geschichte der Plattform und Programmiersprache
 - Anfang 1996: JDK 1.0
 - Anfang 1997: JDK 1.1
 - Ende 1998: J2SE 1.2
 - Swing, Collections
 - Mitte 2000: J2SE 1.3
 - JNDI, RMI, Sound
 - Anfang 2002: J2SE 1.4
 - 64-bit Architektur, NIO, reguläre Ausdrücke, assertions, Schnittstelle für XML-Parser
 - Mitte 2004: J2SE 5.0
 - Generische Klassen, typsichere Aufzählungen, erweiterte for-Schleife, Boxing und Unboxing
 - Ende 2006: Java SE 6
 - Mitte 2011: Java SE 7

- Geschichte der Plattform und Programmiersprache (Fortsetzung)
 - März 2014: **Java SE 8 (long term support, LTS)**
 - Anonyme Funktionen, Closures, Lambda-Funktionen, ...
 - September 2018: **Java SE 11 (LTS)**
 - März 2019: Java SE 12
 - September 2019: Java SE 13
 - März 2020: Java SE 14
 - September 2020: Java SE 15
 - März 2021: Java SE 16
 - September 2021: **Java SE 17 (LTS)**
 - März 2022: Java SE 18
 - September 2022: Java SE 19
 - März 2023: Java SE 20
 - September 2023: Java SE 21
- Wir verwenden für die Übungen **mindestens Java 8**

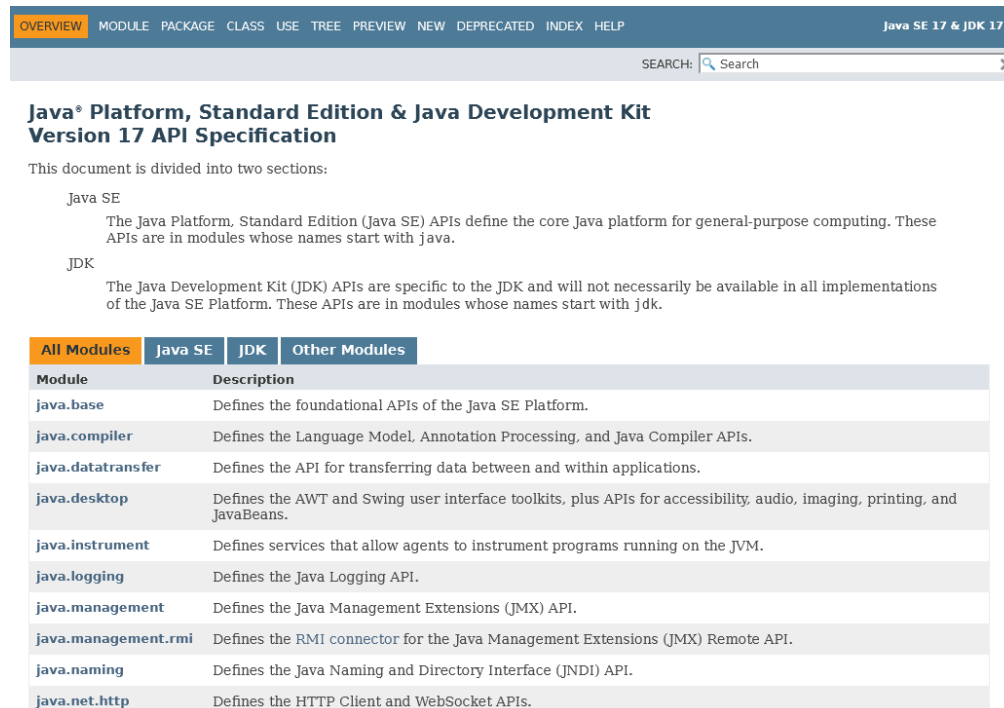
- Enthält Klassen
 - Ausgelegt für Bedürfnisse von Desktop-Anwendungen
 - Klassen sind in Pakete gruppiert
 - Anzahl Klassen und damit Funktionalität ist geringer als J2EE aber umfangreicher als J2ME
- Enthält Anwendungen
 - javac(.exe) (SDK)
 - java(.exe) (SDK und JRE)
 - jar(.exe)
 - javadoc(.exe)
 - ... siehe Verzeichnis <JAVA_HOME>/bin
- Welche Pakete gibt es?
 - java.applet
 - java.awt
 - java.io
 - java.lang
 - java.math
 - java.net
 - java.rmi
 - java.sql
 - java.util
 - javax.swing
 - ...

- Download über Oracle
 - <https://www.oracle.com/de/java/technologies/java-se-glance.html>
 - Versionen für Linux, Mac und Windows
- Nach Installation oft neustart erforderlich
- Überprüfung der Version über Konsole (cmd)
 - „java -version“

```
C:\>java --version
java 17.0.8 2023-07-18 LTS
Java(TM) SE Runtime Environment (build 17.0.8+9-LTS-211)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.8+9-LTS-211, mixed mode, sharing)
```

- Falls Befehl nicht ausgeführt wird -> Umgebungsvariable in Windows überprüfen

- Dokumentation der Klassenbibliothek
 - „API Specification“
 - Online lesbar: <https://docs.oracle.com/en/java/javase/17/>
 - Offline nutzbar (z.B. durch Quick Documentation)



The screenshot shows the Oracle Java SE 17 API Specification documentation page. The page has a navigation bar with tabs: OVERVIEW, MODULE, PACKAGE, CLASS, USE, TREE, PREVIEW, NEW, DEPRECATED, INDEX, and HELP. The current tab is OVERVIEW. Below the navigation bar is a search bar with the text "SEARCH: Search". The main content area is titled "Java® Platform, Standard Edition & Java Development Kit Version 17 API Specification". It states that the document is divided into two sections: Java SE and JDK. Under Java SE, it says: "The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with java." Under JDK, it says: "The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with jdk." Below this text is a table with the following structure:

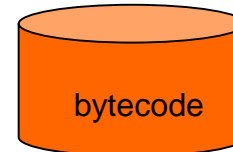
Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.
java.logging	Defines the Java Logging API.
java.management	Defines the Java Management Extensions (JMX) API.
java.management.rmi	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
java.naming	Defines the Java Naming and Directory Interface (JNDI) API.
java.net.http	Defines the HTTP Client and WebSocket APIs.

Die erste Java Anwendung

- Entwicklungsschritte
 - Programmieren
 - Kompilieren mit javac(.exe)
 - Ausführen mit java(.exe)
- Optional
 - Generieren einer Dokumentation mit javadoc(.exe)
 - Verpacken mit jar(.exe)

```
public class HelloWorld {  
    public static void main(String[] args)  
    {  
        System.out.println("Hello World!");  
    }  
}
```

```
>javac HelloWorld.java
```



HelloWorld.class

write once

```
>java HelloWorld  
Hello World!
```

run anywhere

■ English

- package
- class
- interface
- access modifier
- conditional
- loop
- exception
- application
- source code
- slash and backslash
- overriding
- overloading
- runtime

■ Deutsch

- Paket (von Klassen)
- Klasse
- Schnittstelle
- Zugriffs-Modifizierer
- Fallunterscheidung
- Schleife/Wiederholung
- Ausnahme
- Anwendung
- Quellcode
- Schrägstriche (/ und \)
- überschreiben
- überladen
- Laufzeit

- Bildet Objekte der realen Welt ab
- **Objekte** bzw. Instanzen
 - reagieren nur auf Botschaften, die sie verstehen
 - vereinigen Daten und Methoden
- **Methoden**
 - Botschaften werden durch sogenannte Methoden realisiert, d.h. durch Aufruf einer Methode sendet man eine Botschaft
 - Methoden sind Prozeduren sehr ähnlich
- **Klassen** bzw. Objekttypen
 - Objekte, die dieselben Botschaften verstehen und dieselben Merkmale besitzen, bilden eine Klasse von Objekten

■ Kapselung

- Interna eines Objekts werden unsichtbar nach außen gemacht bzw. gekapselt

■ Vererbung

- Enger Zusammenhang mit Wiederverwendbarkeit und Anpassungsfähigkeit
- Neues Objekt wird von vorhandenem Objekt abgeleitet
- Das neue Objekt erhält dann neue charakteristische Merkmale

■ Überschreiben

- Verschiedene Unterklassen verstehen dieselbe Botschaft, obwohl die technische Umsetzung der Reaktion völlig unterschiedlich sein kann
→ Polymorphie (Methoden können „viele Formen“ haben.)
- z.B. `toString()`

■ Überladen

- Klasse stellt verschiedene Varianten derselben Methode zur Verfügung, d.h. unterschiedliche Parameterstruktur
- z.B. `add(int, int)` und `add(int, int, int)`

■ **Klassen**

- public und default
- abstract
- die Schnittstellen implementieren
- die von einer anderen Klasse erben
 - Ausnahmeklassen
 - Keine Mehrfachvererbung
- Innere Klasse (inner, anonymous)
- Variablen einer Klasse
 - Klassen-Variablen (static)
 - Instanz-Variablen
 - Variablen in Operationen der Klasse
- Operationen einer Klasse
 - Klassen-Operationen (static)
 - Konstruktoren bei Klassen
 - Instanz-Operationen
- Feld (Array) von Klassentyp

■ **Pakete**

- Verzeichnisse

■ **Importe**

- Bibliotheken

■ **Schnittstellen** (Interfaces)

- Operationen von Schnittstellenklassen
- Feld (Array) von Schnittstellentyp

■ **Sichtbarkeit**

■ **Typkonvertierung**

- Casting
- z.B. int -> long

Nicht-objektorientierte Sprachelemente

- Primitive Datentypen
 - byte
 - short
 - int
 - long
 - float
 - double
 - char
- Feld (Array) von primitiven Datentypen
- Operatoren
 - +, -, *, /, ...
 - ++, --
 - ==, !=, ...
- Blöcke
 - geschweifte Klammern
- Fallunterscheidungen
 - if, switch
- Schleifen
 - for, while, do while

Unterschiede zwischen C++ und Java

- In Java keine Mehrfachvererbung
- In Java keine private Vererbung
- Java hat keinen eigenständigen Präprozessor → kein #defines oder macros
- Bei Java ist kein Linkprozess notwendig
- Java kennt keine Zeiger → dafür Referenzen zu Objekten
- In Java werden Objekte immer mit new erzeugt
- Speicherverwaltung in Java ist automatisch → dennoch Speicherverbrauch berücksichtigen
- Java kennt keinen Destruktor → Methode `finalize()`

Unterschiede zwischen C++ und Java

- Felder in Java sind durchnummerierte Listen von Referenzen
 - Ausnahme bei Aufrufen eines nicht existierenden Index
 - mehrdimensionale Felder sind Listen von Feldern
- In Java keine Definition von Operatoren möglich (Operatorenüberladung in C++)
- Copy-Konstruktor ersetzt durch Operation `clone()`
- In Java kein `struct`, `union`, `typedef` statement → man deklariert neue Klasse
- Java kennt keine header files → `import` Anweisungen und Interface-Definition

Unterschiede zwischen C++ und Java

- Java kennt kein Schlüsselwort `virtual` → Schnittstelle (Interface)
- Java kennt keine reinen virtuellen Operationen → Schlüsselwort `abstract`
- Java kennt keine globalen Variablen → ersetzen durch `static`
- In Java kein Schlüsselwort `const` → Schlüsselwort `final`
- Java bietet kein `goto` (Sprünge sind z.B. mit `continue` möglich)

■ Teil 1: Überblick

- Erstkontakt
- Java Standard Edition
- Objektorientierte Programmierung (OOP)
- Unterschiede zwischen C++ und Java

■ Teil 2: Java Basics

- Klassen
- Felder (Arrays)
- Methoden
- Konstruktoren
- Vererbung
- Objekte
- Ausnahmen
- Collections, Generizität
- Callbacks, Lambda, Methoden-Referenzen, ...

Cutting corners to meet arbitrary management deadlines



Essential

Copying and Pasting
from

O'REILLY®

The Practical Developer
@ThePracticalDev

365



Inside a method that returns `IEnumerable<T>`, `yield return` has to return `T`, not an `IEnumerable<T>`.

Replace

```
yield return c.GetDeepControlsByType<T>();
```

with:

```
foreach (var x in c.GetDeepControlsByType<T>())
{
    yield return x;
}
```

[share](#) [improve this answer](#)

answered Jan 13 '10 at 10:29



Marcin Seredynski
4,719 ● 2 ● 14 ● 22

389 I just had exactly the same problem and googled this solution. I used it. I wanted to thank the author and got "You can't vote for your own post". I laughed when I realized what has just happened. Too much work kills your brain cells... – [Marcin Seredynski](#) Sep 12 '11 at 19:24

'14 at 12:35

6 [@KMX](#) - his vOcation is obviously 'Software Developer'! – [FastAI](#) Oct 6 '14 at 13:43

4 [@MarcinSeredynski](#) - This is without a doubt the best comment I've ever seen and is a poignant reminder of the nature of software development. – [bubbleking](#) Dec 29 '15 at 20:33

[show 4 more comments](#)

■ Aufbau einer Klasse

```
package <PaketName>;

import <PaketOderEinzel>;

<modifiers> class <KlassenName>
    extends <KlassenName>
    implements <InterfaceName>, ...
{
    <Abschnitt Variablen-Definitionen>

    <Abschnitt Konstruktoren-Definitionen>

    <Abschnitt Operationen-Definitionen>
}
```

■ Beispiel

```
package com.my.graphics;

import java.lang.Object;
import java.util.*;

public class Point {
    public int x, y, color;

    public void moveTo(int newX, int newY) {
        x = newX;
        y = newY;
    }

    public void moveRel(int dx, int dy) {
        x += dx;
        y += dy;
    }
}
```

■ Schnittstellen

- Kennzeichen ist Schlüsselwort `interface`
- Instanzmethoden dürfen nur `public` und `default` als Zugriffs-Modifizier besitzen
- Instanzmethoden werden normalerweise nicht implementiert (außer `default`-Methoden)
- Klassenvariablen kompilieren, können aber nicht von der implementierenden Klasse geändert werden => Konstanten
- Können Klassenmethoden (=statische Methoden) enthalten

■ Beispiel

```
public interface Person {  
    public String sayHello();  
    Integer tellAge() throws Exception;  
}  
  
public interface RadfahrerIn {  
    public void beschleunige();  
}
```


- Wozu Schnittstellen?
- Was ist, wenn man die Operationen weglässt?
- Implementieren einer Schnittstellenklasse

```
public class Myself implements Person, RadfahrerIn {  
  
    // has to be implemented  
    public void sayHello() {  
        // implementation goes here  
    }  
  
    // has to be implemented  
    Integer tellAge() throws Exception {  
        // implementation goes here  
    }  
    // Methods from RadfahrerIn  
    // go here  
}
```

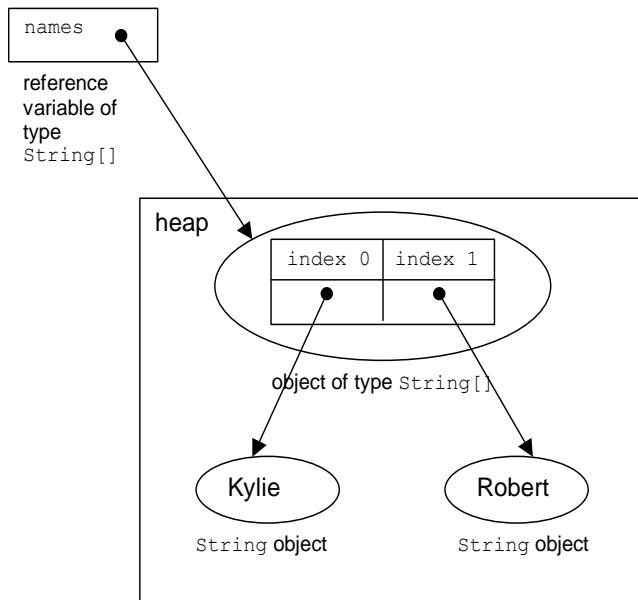
■ Innere Klasse

- Werden innerhalb einer äußeren Klasse deklariert und stehen nur dieser zur Verfügung
- Werden verwendet, wenn eine Hilfsklasse gewünscht wird und diese Kenntnis von den Attributen der äußeren Klasse benötigt

```
public class ClassWithInnerClass {  
    private Vector items;  
  
    public Enumeration enumerator() {  
        return new MyStackEnum();  
    }  
  
    // inner class start  
    class MyStackEnum implements Enumeration  
    {  
        int currentItem = items.size() - 1;  
        public boolean hasMoreElements() {  
            return (currentItem >= 0);  
        }  
        public Object nextElement() {  
            if (!hasMoreElements()) {  
                throw new NoSuchElementException();  
            } else {  
                return items.elementAt(currentItem--);  
            } // end if  
        } // end method  
    } // inner class end  
  
} // end of outer class
```

- **Felder von Klassen und von primitiven Datentypen**
 - sind selbst Objekte, auch wenn die Elemente primitive Typen (wie `int`) sind
 - Drei Dinge sind notwendig um Felder zu füllen:
 - Die Deklaration einer Feld-Referenz-Variable
 - Das Konstruieren eines Feld-Objekts
 - Das Belegen der Elemente des Feld-Objekts

■ Visualisierung



```
public class ArrayTester {
    int[] key; // array of primitives
    static String[] names; // array of objects
    static int[] scores = new int[10];
    // or in shortcut notation
    String[] chiles = {"jalapeno", "serrano"};
    int x = 9;
    int[] yArray = {3, 6, x, 12};

    public static void main(String[] args) {
        names = new String[2];
        // Werte zuweisen
        names[0] = "Kylie";
        names[1] = "Robert";
        // Werte zuweisen
        for (int i=0; i<5; i++) {
            scores[i] = i;
        }
    }
}
```

■ **package**

- Die Klasse oder die Schnittstelle einer Datei gehören zu einem Paket
- Unterlässt man die Definition, befindet sich die Klasse oder die Schnittstellenklasse im Paket „default“
- Die Definition hat Auswirkung auf die Sichtbarkeit einer Klasse für Klassen in demselben oder einem anderen Paket
- Kategorisierung
- Zusammenhang mit Verzeichnisstruktur

■ **import**

- von einzelnen Klassen
- von einzelnen Schnittstellen
- von ganzen Paketen
- CLASSPATH

■ **public**

- einer der 3 access modifier

■ **class** oder **interface**

■ **extends**

- erbt von Klasse

■ **implements**

- implementiert Schnittstellenklasse

■ Dateiname

- Der Dateiname muss gleich dem Namen der öffentlichen Klasse sein
- Aus öffentlicher Klasse Point wird Datei Point.java
- Keine weitere public Klasse, es können aber noch default Klassen in der Datei enthalten sein
- Kann ein Interface oder sogar mehrere mit in der Datei sein?

■ Verzeichnis

- Die Teile des Pakets ergeben die Namen der hierarchisch angeordneten Verzeichnisse
- Aus Paket com.my.graphics wird Verzeichnisstruktur
<Irgendwo>/com/my/graphics

■ Vier Möglichkeiten

- `//` Einzeiler oder am Zeilenende
- `/* ... */` Einzeiler oder am Zeilenende
- `/* ... * ... */` Mehrzeiler
- `/** ... * ... */` javadoc

```
/**
 * Prints out a welcome message
 * @version 1.0
 */
class HelloWorld {
    /*
     * Say Hello
     */
    public static void main(String[] args) {
        // output text
        System.out.println("Hello World!");
    }
}
```

■ Javadoc Tags:

- `@author`
 - classes and interfaces only, required
- `@version`
 - classes and interfaces only, required
- `@param`
 - methods and constructors only
- `@return`
 - methods only
- `@exception`
- `@throws`
- `@see`
- `@since`
- `@deprecated`

■ Variablen

- zu verwenden nur bei Klassen, nicht bei Schnittstellen
- gehören entweder einer Instanz → Instanzvariable
- oder einer Klasse → Klassenvariable bzw. statische Variable (modifier static)
- oder befinden sich innerhalb einer Methode → lokale Variable
- Instanzvariablen und Klassenvariablen werden auch Members oder Attribute genannt

```
public class APerson {  
    // Beispiel für?  
    public String lastName = "";  
    private String firstName;  
  
    // Beispiel für?  
    public static final boolean  
        eachPersonHasAName = true;  
    private static int numLegs = 2;  
  
    public void doSomething() {  
        int i=5;  
    }  
}
```

■ Ein Wort zu Attributen

- Immer mit getX- und setX- Methoden kapseln
- Kapselung: Einschränkung des Zugriffs
- „Getters and Setters“ oder „Accessors and Mutators“

■ Operationen

- zu verwenden bei Klassen und Schnittstellen

```
<modifiers> <return type> <MethodenName>  
([Parameter1, Parameter2, ...])  
[throws Exception1, ...] {  
    // Methodenkörper  
}
```

■ Beispiel

```
public void doSomething() {  
    // implement here  
}  
  
public Integer calculate(Integer  
    i1, String s1) throws Exception  
{  
    // implement here  
}
```

- Was tut man hier?

```
public class DoClass {  
  
    public void doSomething(Integer i) {  
    }  
  
    // OK?  
    public void doSomething(String s) {  
    }  
  
    // OK?  
    public String doSomething(Integer i) {  
    }  
  
    // OK?  
    public String doSomething(int i) {  
    }  
  
} // end of class
```

- Ist das erlaubt?

```
public class ClassWhatsThis {  
  
    // Methode?  
    public ClassWhatsThis() { }  
  
} // end of class
```

- Aufgerufen beim Erzeugen eines Objekts => nur bei Klassen
- Überladen möglich
- Im Unterschied zu Methoden haben Konstruktoren keinen Ergebnistyp
- Jede Klasse, die keinen Konstruktor implementiert erhält einen parameterlosen Default-Konstruktor
- Es ist möglich den Default-Konstruktor zu überschreiben
- derselbe Name wie Klasse
- Mehrere Konstruktoren mit unterschiedlichen Parametern sind möglich
 - Wozu?

```
// eigener default constructor  
public <KlassenName> { }
```

- Welchen Konstruktor hat die folgende Klasse?

```
public class ClassWithoutConstructor {  
}
```

- Warum ist es nicht möglich zu kompilieren?

```
public class DoesNotCompile {  
    public String DoesNotCompile(int i) {  
    }  
}
```

- Wieviele Konstruktoren hat diese Klasse?

```
public class ClassHowMany {  
    public ClassHowMany(String s) {  
    }  
}
```

- Was passiert hier mit dem Konstruktor?

```
public class ClassTwoCons {  
    public ClassTwoCons(Integer i) {  
    }  
  
    public ClassTwoCons(String s) {  
        this(new Integer(5));  
    }  
}
```

Fragen?

Michael Anders

Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 205
69120 Heidelberg, Germany
<https://se.ifi.uni-heidelberg.de>



RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG
