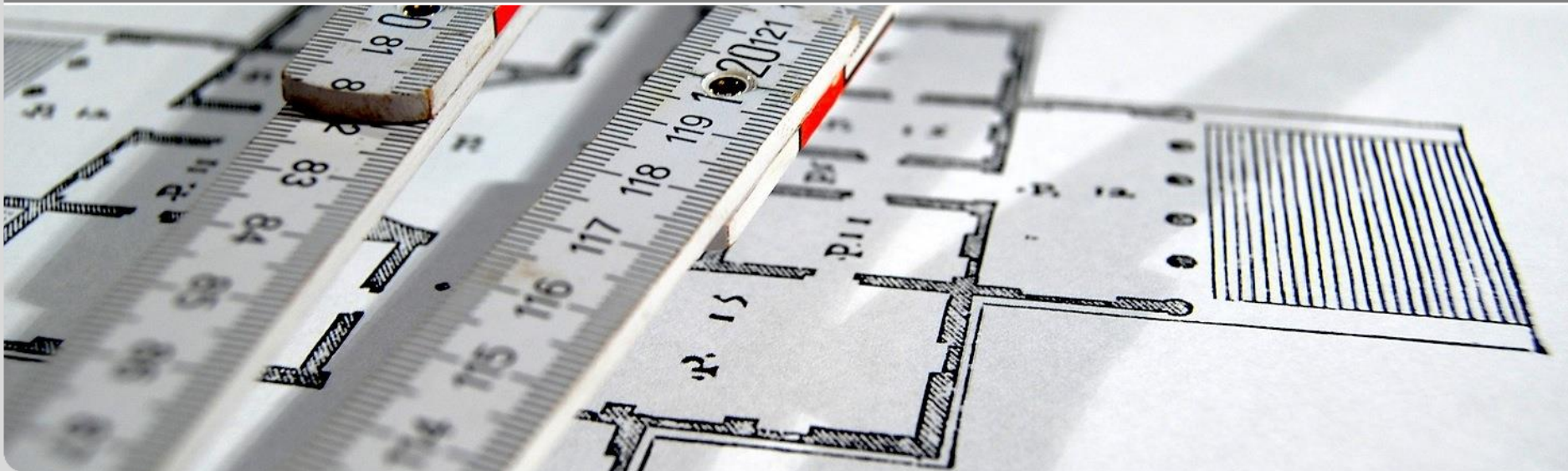


Programmieren-Tutorium Nr. 10

5. Tutorium | Jonas Ludwig
Eclipse, Datenkapselung

Architecture-driven Requirements Engineering – Institut für Programmstrukturen und Datenorganisation – Fakultät für Informatik



Was machen wir Heute?

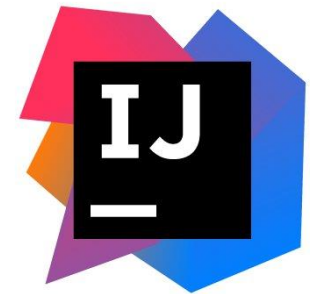
- Korrektur ÜB1
- Entwicklungsumgebungen
 - IntelliJ IDEA / Eclipse
 - Checkstyle
 - Terminal.java
- Pakete
- Datenkapselung
- toString()-Methode
- Arrays
- Kommandozeilenargumente

Übungsblatt 2

- Aufgaben genau lesen!
- Checkstyle verwenden!
- Terminal-Klasse verwenden, aber nicht mit abgeben!

IntelliJ IDEA

- Integrierte Entwicklungsumgebung zur Programmierung von Software in Java, Kotlin, Groovy und Scala
 - Aktuelle Version: 2019.2.3 (24.11.2019)
 - Zwei Versionen
 - Community Edition
 - Generell kostenlos
 - eingeschränkte Funktionalität
 - Ultimate Edition
 - Für Studenten kostenlos
 - Zuerst: <https://www.jetbrains.com/de-de/student/>
 - Dann: <https://www.jetbrains.com/de-de/idea/download/>
 - Volle Funktionalität
- Weitere Java Entwicklungsumgebungen
 - Eclipse
 - NetBeans IDE

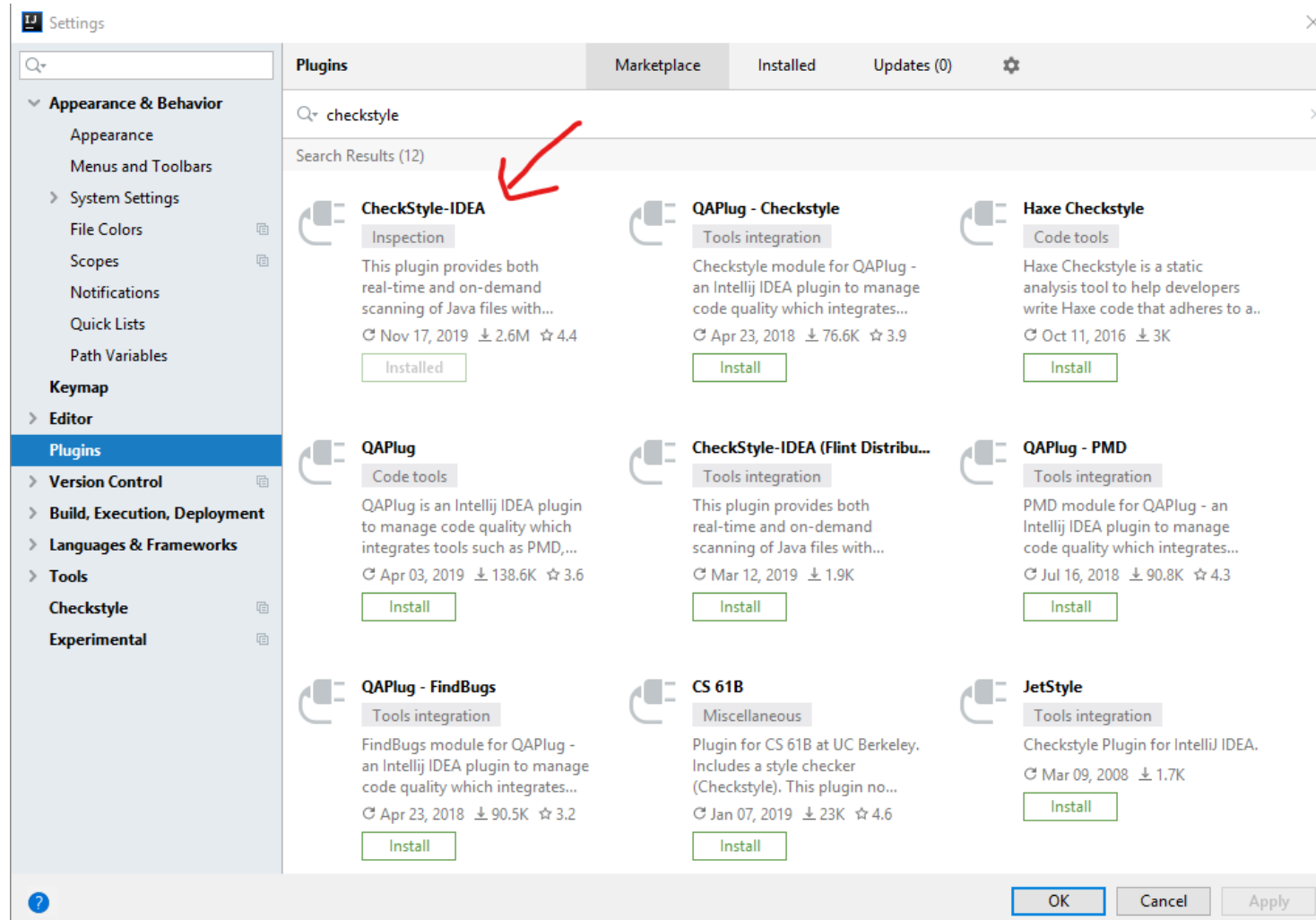


IntelliJ IDEA - Livedemo

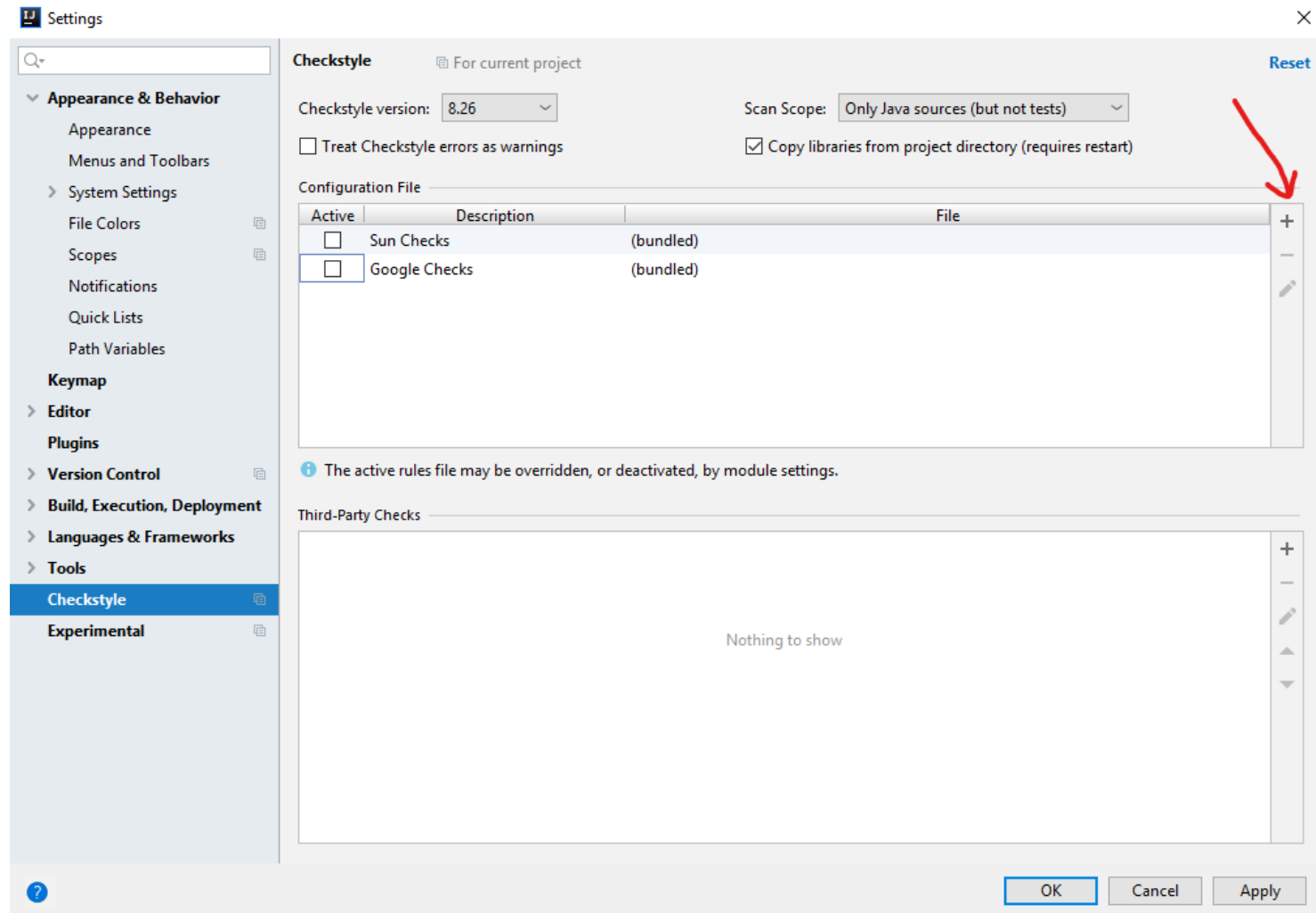
Zum Nachlesen/Nachschlagen

- IntelliJ IDEA installieren
 - Account anlegen: <https://www.jetbrains.com/de-de/student/>
 - Download: <https://www.jetbrains.com/de-de/idea/download/>
 - Ultimate Edition für Studenten kostenlos
- Checkstyle installieren
 - Einstellungen > Plugins > Nach „CheckStyle“ suchen
> „CheckStyle-IDEA“ installieren > IntelliJ IDEA neustarten
- Checkstyle importieren
 - Einstellungen > Checkstyle > oben rechts auf „+“ > Checkstyle auswählen
> „Aktiv“ auswählen
- Checkstyle-konforme Einrichtung: Leerzeichen statt Tabulatoren
 - https://ilias.studium.kit.edu/goto.php?target=wiki_851245_Eclipse_IDE
- Nächste Woche mehr zu Checkstyle/JavaDoc...

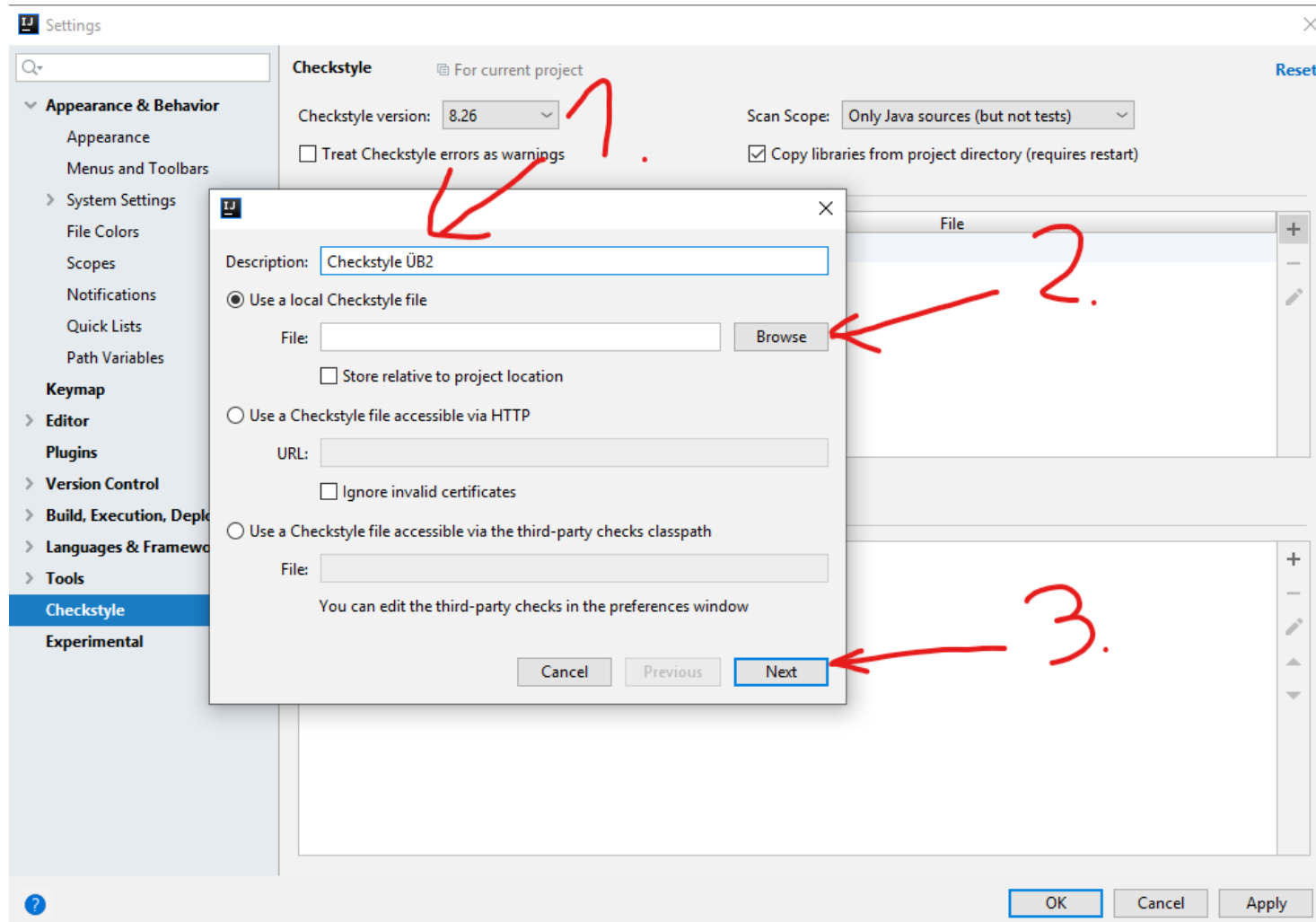
Checkstyle installieren



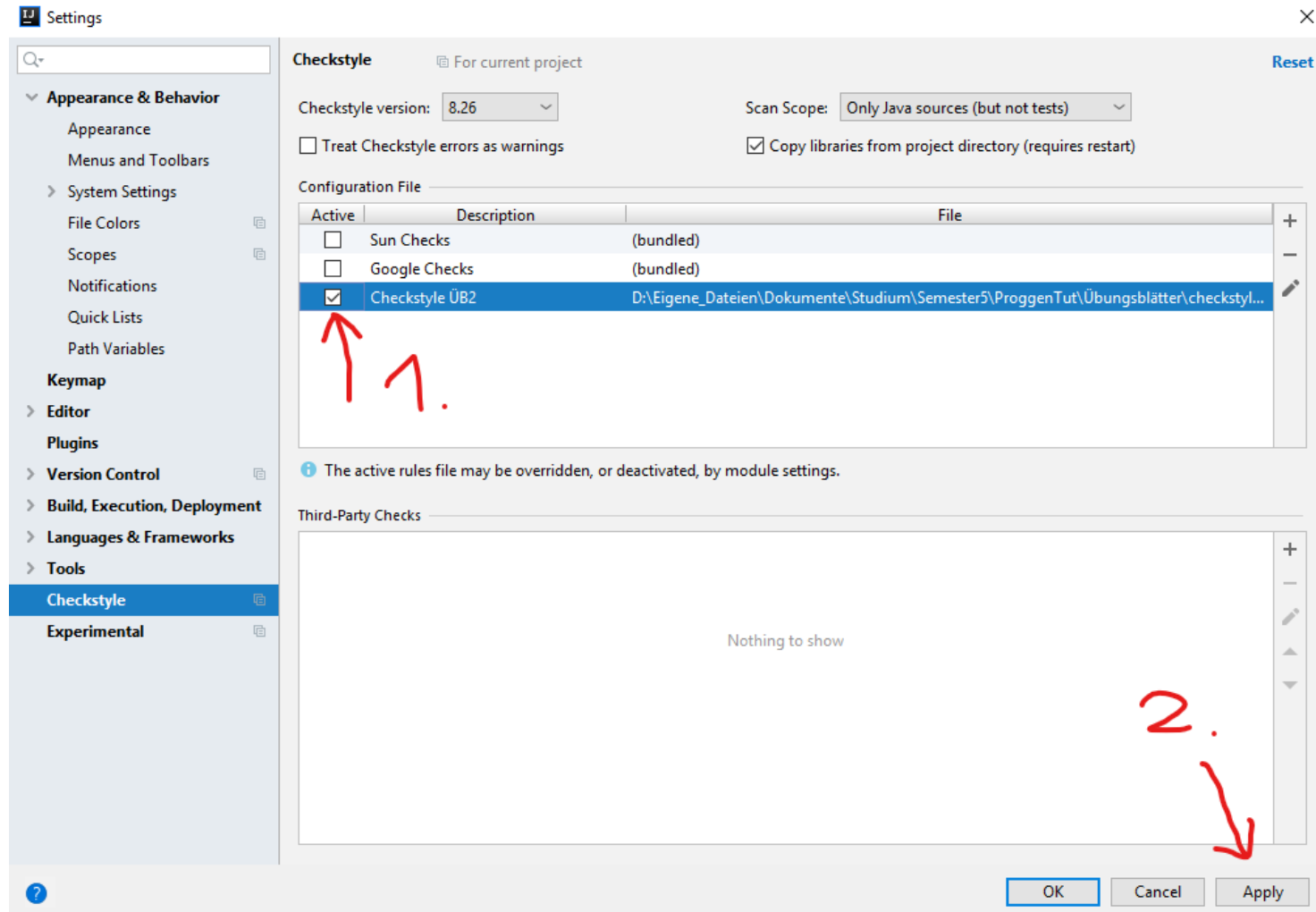
Checkstyle laden



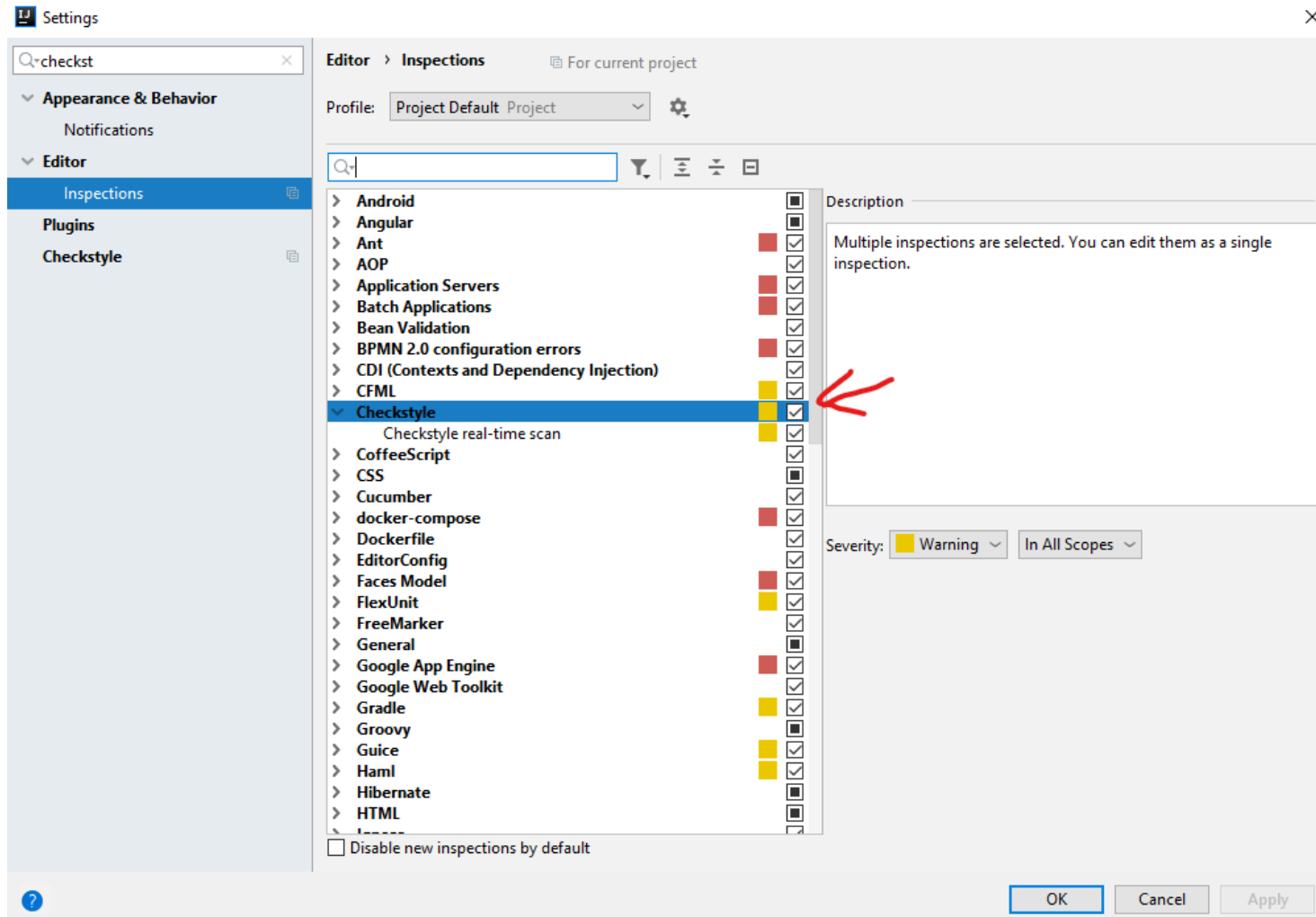
Checkstyle laden



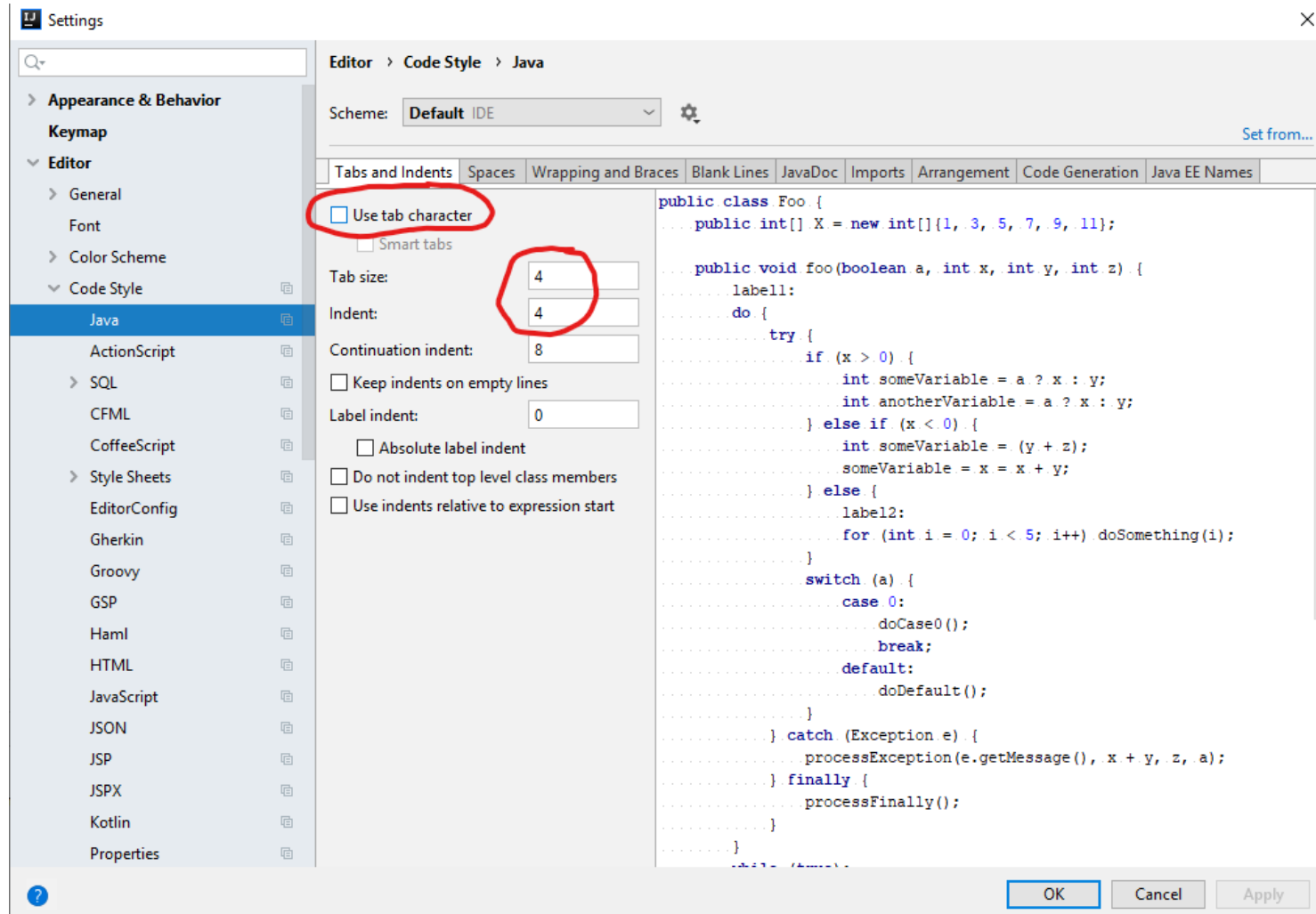
Checkstyle laden



Checkstyle Real-Time Scan aktiv?



Checkstyle – Einrückung mit 4 Leerzeichen



Alternativ: Eclipse

- Ein auf Java basierende integrierte Entwicklungsumgebung zur Programmierung von Software
 - Aktuelle Version: 4.9 (2018-09)
 - <https://www.eclipse.org/downloads/>

- Weitere Java Entwicklungsumgebungen
 - IntelliJ IDEA (als Student kostenlos)
 - NetBeans IDE



Alternativ: Eclipse

Zum Nachlesen/Nachschlagen

- Eclipse installieren
 - <https://www.eclipse.org/downloads/packages/>
 - “Eclipse IDE for Java Developers”
- Checkstyle installieren
 - https://ilias.studium.kit.edu/goto.php?target=wiki_851245_Checkstyle
- Checkstyle-konforme Einrichtung: Leerzeichen statt Tabulatoren
 - https://ilias.studium.kit.edu/goto.php?target=wiki_851245_Eclipse_IDE
- Nächste Woche mehr zu Checkstyle/JavaDoc...

Terminal-Klasse

- Für die Aus-/Eingabe verwenden!

```
Terminal.println(/*Some String*/);
```

```
Terminal.printError(/*Some String*/)
```

```
Terminal.readLine()
```

```
Terminal.readFile(/*Some File*/)
```

- Klasse muss in folgendem Paket liegen:

```
edu.kit.informatik
```

- Klasse darf nicht mit im Praktomat abgegeben werden!

Pakete

- Pakete in Java dienen der Strukturierung des Quelltextes und sollen Namenskonflikte vermeiden
- Innerhalb eines Pakets ist jeder Name eine Klasse eindeutig
 - Der gleiche Name kann jedoch in einem anderen Paket frei zur Bezeichnung einer anderen Klasse benutzt werden

```
package java.awt;
```

```
public class Date {  
    //...  
}
```

```
package java.sql;
```

```
public class Date {  
    //...  
}
```

- Eine Klasse ist Teil eines Paketes, wenn:
 - Sie in dem Ordner mit dem Paketnamen liegt
 - Sie `package [paketname] ;` ganz am Anfang stehen hat

Pakete

- Der Namen von Paketen in Java wird immer klein geschrieben!
- Pakete können selber wieder beliebig viele Unterpakete haben:
 - `java.lang.String`
 - `java.lang.Math`
 - `java.lang.annotation.RetentionPolicy`
 - `java.lang.invoke.MethodType`
- Beispielhafter `java.lang` Auszug:



Import von Paketen

- Um eine Klasse verwenden zu können, muss angegeben werden, in welchem Paket sie liegt. Hierzu gibt es zwei unterschiedliche Möglichkeiten:
- Die Klasse wird über ihren vollen (qualifizierten) Namen angesprochen:
 - `java.awt.Date d = new java.awt.Date();`
- Am Anfang des Programms werden die gewünschten Klassen mit Hilfe einer import-Anweisung eingebunden:
 - `import java.awt.Date;`
`//...`
`Date d = new Date();`
- Alle Klassen im Paket `java.lang` sind für die Sprache so essentiell, dass sie von jeder Klasse automatisch importiert werden

Import von Paketen

- Bei den import-Anweisungen sollten keine Wildcards verwendet werden
 - Schafft Übersicht und macht es für andere Softwareentwickler einfacher zu verstehen welche Klasse genau gemeint und gebraucht ist
- Das Importieren von nicht benötigten Klassen kann den eigenen Namensraum stören, da es zu Namenskonflikten kommen kann
 - Dies kann auch passieren, wenn nachträglich in ein importiertes Paket eine neue Klasse hinzugefügt wird!

```
import java.awt.*;  
import java.util.*;  
  
class TheBigBang {  
    List someList = new List();  
}
```

Datenkapselung

- Attribute werden vor dem Zugriff von außen verborgen
 - Der direkte Zugriff auf die interne Datenstruktur wird unterbunden und erfolgt stattdessen über definierte Methoden
 - Objekte können den internen Zustand anderer Objekte nicht in unerwarteter Weise lesen oder ändern
- Abschotten der internen Implementierung vor direktem externem Zugriff
 - Dieser darf nur über eine explizit definierte Schnittstelle erfolgen, um ihn unabhängig von den Implementierungsdetails zu machen
 - Die Implementierung von Klassen kann geändert werden, ohne die Zusammenarbeit mit anderen Klassen zu beeinträchtigen



Zugriffsmodifikatoren

- Mit Zugriffsmodifikatoren lassen sich die Sichtbarkeiten von Programmteilen regeln

	Klasse	Paket	Unterklasse (gleiches Paket)	Unterklasse (anderes Paket)	Welt
public	✓	✓	✓	✓	✓
protected	✓	✓	✓	✓	✗
no modifier	✓	✓	✓	✗	✗
private	✓	✗	✗	✗	✗

Sichtbarkeit für Klassen

```
package mypackage;

public class Point {

    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return this.x;
    }

    public int getY() {
        return this.y;
    }

    public boolean equals(Point other) {
        if (other == null) {
            return false;
        }
        return (this.x == other.x) && (this.y == other.y);
    }
}
```

```
package mypackage;

public class Test {
    public static void main(String[] args) {
        Point myPoint = new Point(1, 1);
        System.out.println(myPoint.getX());
        // The field Point.y is not visible
        System.out.println(myPoint.y);
    }
}
```

Compilerfehler

Ist innerhalb der
Klasse sichtbar

Regeln für Zugriffsmodifikatoren

- Klassen sind in der Regel immer **public**
- Sämtliche Attribute einer Klasse sollten **private** sein
- Bei Konstanten (`static final`) kann **public** sinnvoll sein
- Schnittstellen-Methoden sind **public**
 - Schützen das Klassengeheimnis
 - Bieten Abstraktion über Implementierungsdetails
 - Haben eine Aufgabe, die mit dem Namen zusammenhängt
- Hilfsmethoden sind in der Regel immer **private**
 - Sind funktionale und oder logische Einheiten in sich
 - Vermeiden Code-Redundanz

Sichtbarkeit in Paketen

- Klassen kennen normalerweise nur die Klassen im eigenen Paket
- Falls eine Klasse ohne Paket-Angabe implementiert wird, befindet sie sich standardmäßig im unbenannten Paket
 - Eine im Paket befindliche Klasse kann jede andere sichtbare Klasse aus anderen Paketen importieren, aber keine Klassen aus dem unbenannten Paket
- Kein Zugriffsmodifikator bedeutet, dass ein Element nur innerhalb seiner Klasse und der Klassen im selben Paket sichtbar ist.
 - Verhält sich innerhalb eines Paketes wie: **public**
 - Verhält sich außerhalb eines Paketes wie: **private**

toString-Methode

- Jedes Objekt sollte sich durch die Methode **toString()** mit einer Zeichenkette identifizieren

- Inhalt der interessanten Attribute als Zeichenkette liefern

```
public class Player {  
    String name;  
    int age;  
  
    public String toString() {  
        return "[name=" + name + ", age=" + age + "];"  
    }  
}
```

- Die Methode wird automatisch aufgerufen, wenn die Methoden **print()** oder **println()** mit einer Objektreferenz aufgerufen werden
 - Ähnliches gilt für den Zeichenkettenoperator **+** mit einer Objektreferenz als Operand

Signatur der toString-Methode

- Die toString-Methode muss immer mit dem Zugriffsmodifizierer `public` versehen sein!

```
public String toString() {  
    //Rumpf  
}
```

Array Grundidee

- Oft benötigt man nicht nur Einzelwerte (eine Variable) sondern eine Wertemenge, die gemeinsam verarbeitet werden können
- Diese Wertmengen werden oft in Tabellen oder Listen angeordnet
- Ein Array ist eine Liste von Elementen mit dem gleichen Datentyp

```
class BikeShop {  
    Bike bike1;  
    Bike bike2;  
    Bike bike3;  
    Bike bike4;  
    Bike bike5;  
    //...  
}
```

Array Grundlagen

- Ein Array hat einen Namen: `a`
 - Die einzelnen Elemente verhalten sich wie namenlose Variablen
- Ein Array hat eine feste Größe: `n`
 - Die Größe eines Arrays wird einmal festgelegt und ist dann fix
- Das erste Element im Array hat den Index: `[0]`
- Angesprochen werden die einzelnen Elemente durch ihren Index: `([0], [1], [2], ..., [n-1])`

Erzeugung eines Arrays

■ Deklaration des Namens der Arrayvariable:

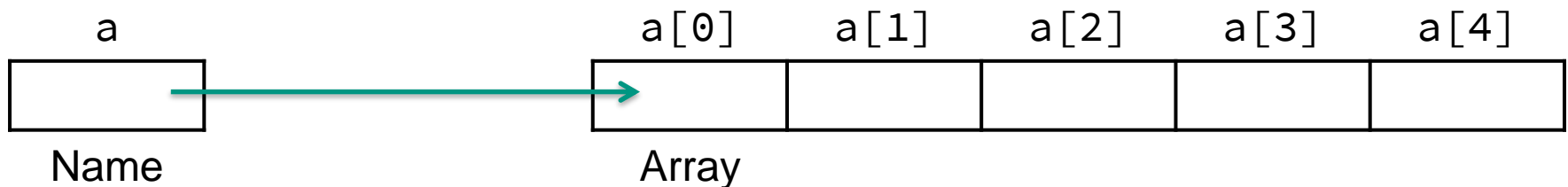
- `int[] a;`
- `double[] vector;`

■ Erzeugung des Arrays:

- `a = new int[5];` // Array mit Index 0 bis 4
- `vector = new double[3];` // 3D Vektor


■ Arrays können schon bei der Deklaration initialisiert werden:

- `char[] abc = {'a', 'b', 'c'};`
- `double[] vector = new double[3];`




Array Grundlagen

```
class Date {  
    /*  
    * date[0] = day  
    * date[1] = month  
    * date[2] = year  
    */  
    int[] date;  
}
```



Ein Array macht hier
wenig Sinn!

```
class BikeShop {  
    Bike[] bikes;  
    //...  
}
```



Bezeichner für Arrays wenn
möglich im Plural!

Array Deklaration

```
byte[] rowvector, colvector, matrix[];  
short key, keyPairs[];  
int a[];
```



So nicht!

■ Was für eine Datentyp ist ... genau?

- rowvector = byte[]
- colvector = byte[]
- matrix = byte[][]
- key = short
- keyPairs = short[]
- a = int[]

■ Bei der Deklaration eines Arrays, kommen die Klammern immer direkt nach der Typdeklaration! Sie ist Teil des Typs und nicht des Namens.

Benutzung eines Arrays

- Der Index muss immer von dem Datentyp Integer sein.
 - Die Länge von einem Array ist auf `Integer.MAX_VALUE` beschränkt.
- Arrayelemente werden durch den Index angesprochen und können wie normale Variablen verwendet werden
 - ```
int[] array = new int[(int) (Math.random() * 10.0)];
array = new int[4];
int x = 1;
int y = 3;
array[3] = x;
array[2 * x + 1] = array[y];
array[Math.max(y, x)] = 100;
```
- Die Länge eines Arrays bekommt man mit: `arrayvariable.length`
  - `array.length` // liefert den int-Wert 4
- Erster zur Laufzeit wird geprüft ob der verwendete Index gültig ist!

# Benutzung eines Arrays

## ■ Typische Schleife für Arrays:

- `for (int i = 0; i < array.length; i++) { }`

## ■ Beispiel:

```
int[] a = new int[5];
```

```
for (int i = 0; i < a.length; i++) {
 a[i] = (i + 1);
 System.out.println(a[i]);
}
```



# Die erweiterte for-Schleife

- Die erweiterte for-Schleife löst sich vom Index und erfragt jedes Element des Felds
  - `for (element : array) { }`
- Rechts vom Doppelpunkt steht ein Array
- Links wird eine lokale Variable deklariert, die später beim Ablauf jedes Element des Arrays annehmen wird

```
for (int element : array) { }
```

```
for (String value : list) { }
```

- Die Schleife liefert ein Element, kann aber nicht in das Feld schreiben
- Die erweiterte Schleife läuft immer das ganze Feld ab
  - Anfang- und Ende-Index können nicht ausdrücklich gesetzt werden

# Beispiel für die erweiterte for-Schleife

```
int[] array = new int[5];
for (int element : array) {
 element = 2;
}

for (int number : array) {
 System.out.print(number);
}

System.out.println();

for (int i = 0; i < array.length; i++) {
 array[i] = (i + 1);
}

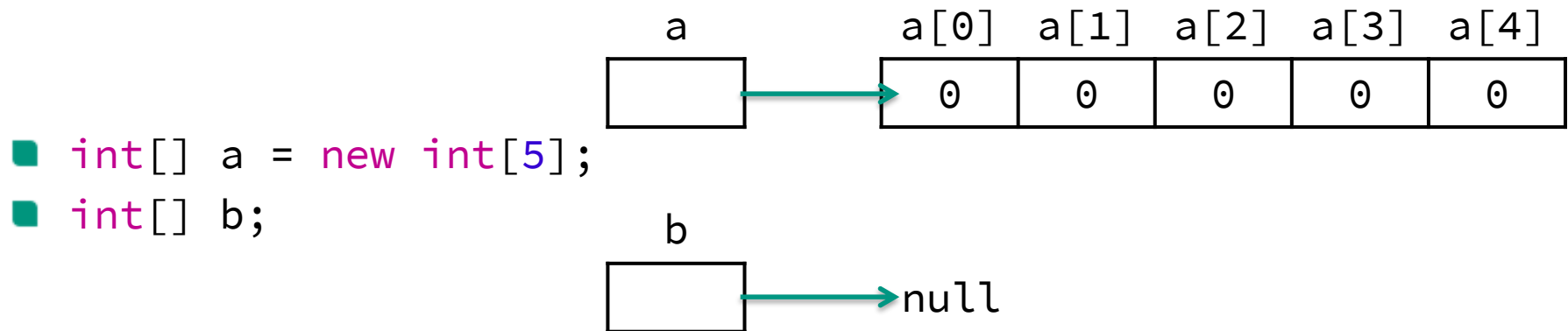
for (int i = 0; i < array.length; i++) {
 System.out.print(array[i]);
}
```

■ Ausgabe:

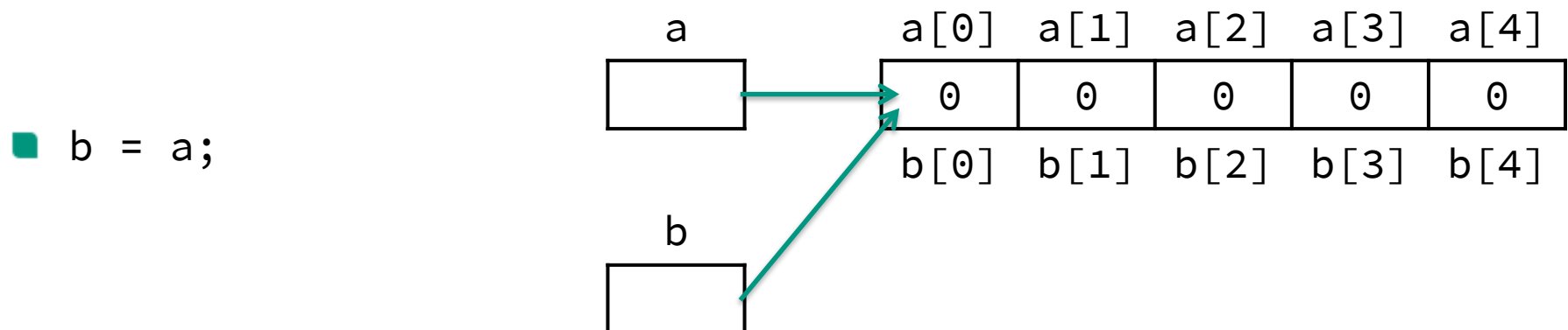
- 00000
- 12345

# Zuweisung eines Arrays

- Einer Arrayvariable dürfen alle Arrays des passenden Datentyps zugewiesen werden



- Beim „Kopieren“ von Objekten ist allerdings Vorsicht geboten



# Aufgabe

```
String[] array1 = {"Geh", "du", "alter", "Esel", "hole", "Fische"};
String[] array2 = array1;
array2[5] = "Erde";
```

```
System.out.print("Array1: ");
for (int i = 0; i < array1.length; i++) {
 System.out.print(array1[i] + " ");
}
```

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

```
System.out.print("Array2: ");
for (int i = 0; i < array2.length; i++) {
 System.out.print(array2[i] + " ");
}
```

## ■ Ausgabe?

Array1: Geh du alter Esel hole Erde

Array2: Geh du alter Esel hole Erde

# Aufgabe

- Schreiben Sie eine Methode, welche alle Referenzen der String Elemente in ein neu erzeugtes Array kopiert und dieses anschließend zurück gibt.

```
public static String[] copyArray(String[] origin)
 String[] copy = new String[origin.length];
 for (int i = 0; i < origin.length; i++) {
 copy[i] = origin[i];
 }
 return copy;
}
```

# Kopieren eines Arrays

- Zum tatsächlichen Kopieren kann eine der Methoden `System.arraycopy()` verwendet werden
- `public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
  - `src` – Das zu kopierende Quell-Array
  - `srcPos` – Startposition im Quell-Array
  - `dest` – Das Ziel-Array
  - `destPos` – Startposition im Ziel-Array
  - `length` – Anzahl der Array-Elemente, welche kopiert werden sollen

```
int[] a = {1,2,3,4,5};
int[] b = new int[5];
System.arraycopy(a, 0, b, 0, a.length);
```

# Aufgabe

- Schreiben Sie eine Methode, die zurückgibt, ob das übergebenen Arrays instanziiert wurde oder leer ist.
  - `public static boolean isEmpty(int[] array)`
- Schreiben Sie eine Methode, die die Summe der Zahlen des übergebenen Arrays als Rückgabewert hat.
  - `public static long arraySum(int[] array)`
- Schreiben Sie eine Methode, die den durchschnittlichen Wert der Zahlen des übergebenen Arrays als Rückgabewert hat.
  - `public static double average(int[] array)`

# Aufgabe – Lösung

```
public static boolean isEmpty(int[] array) {
 return (array == null) || (array.length == 0);
}

public static long arraySum(int[] array) {
 long ret = 0;

 if (!isEmpty(array)) {
 for (int i = 0; i < array.length; i++) {
 ret += array[i];
 }
 }
 return ret;
}

public static double average(int[] array) {
 double ret = 0.0;

 if (!isEmpty(array)) {
 ret = (double)arraySum(array) / (double)array.length;
 }
 return ret;
}
```



# Mehrdimensionales Array

## ■ Dimensionen bei Array:

### ■ 1D – Liste:

```
int[] liste = new int[10];
liste[5] = 5;
```

### ■ 2D – Tabelle:

```
int[][] tabelle = new int[20][30];
tabelle[1][2] = 25;
```

### ■ 3D – Quader:

```
int[][][] quader = new int[4][5][6];
quader[0][0][0] = 50;
```

## ■ Quadratische Matrix:

```
int[][] matrix = {{ 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }};
int zeilen = matrix.length;
int spalten = matrix[0].length;
```

# Mehrdimensionales Array

```
int[][] triangle;
int height = 5;
```

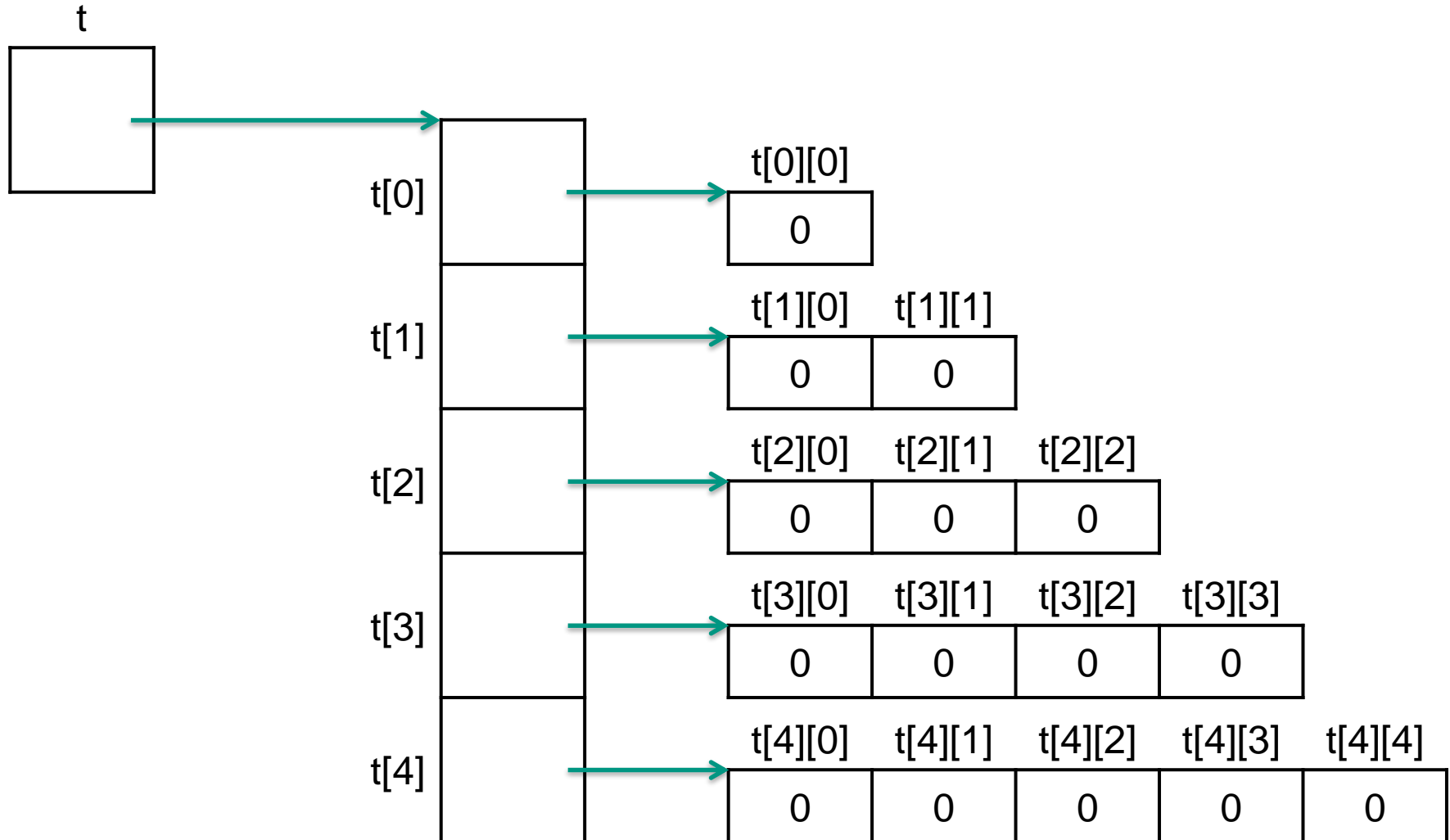
```
//Deklaration
```

```
triangle = new int[height][];
for (int i = 0; i < triangle.length; i++) {
 triangle[i] = new int[i + 1];
}
```

```
//Ausgabe
```

```
for (int i = 0; i < triangle.length; i++) {
 for (int j = 0; j < triangle[i].length; j++) {
 System.out.print(triangle[i][j]);
 }
 System.out.println();
}
```

# Mehrdimensionales Array



- Schreiben Sie eine Methode die die Summe der Zahlen der übergebenen Matrix als Rückgabewert hat. Verwenden Sie dabei keine anderen Methoden.
  - `public static long matrixSum(int[][] matrix)`

# Aufgabe – Lösung

```
public static long matrixSum(int[][] matrix) {
 long ret = 0;

 if (!((matrix == null) || (matrix.length == 0))) {
 for (int i = 0; i < matrix.length; i++) {
 if (!((matrix[i] == null) || (matrix[i].length == 0))) {
 for (int j = 0; j < matrix[i].length; j++) {
 ret += matrix[i][j];
 }
 }
 }
 }

 return ret;
}
```

# Kommandozeilenargumente

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

- Kommandozeilenargumente werden in `String[] args` abgelegt

- Wird also ein Programm aufgerufen mit

```
> java TestApplication Hallo Welt 123
```

Dann ist

- `args.length = 3`
  - `args[0] = "Hallo"`
  - `args[1] = "Welt"`
  - `args[2] = "123"`
- 
- Umwandeln von String zu Zahlen:
  - `Integer.parseInt(String)` bzw.  
`Double.parseDouble(String)`

# Fragen?

# Was machen wir nächste Woche?

- Listen
- Abstrakte Datentypen



# Vielen Danke für eure Aufmerksamkeit!

