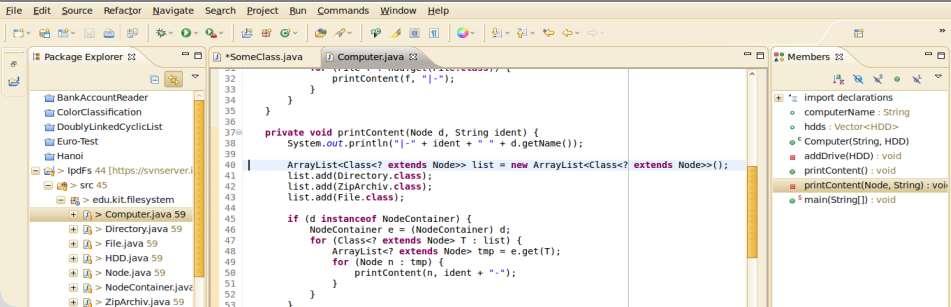


# Programmieren-Tutorium Nr. 10 bei Martin Thoma

Eclipse, Arrays, Kontrollstrukturen und Konventionen

Martin Thoma | 7. November 2012

FAKULTÄT FÜR INFORMATIK



The screenshot shows the Eclipse IDE interface. The Package Explorer on the left lists a project named 'edu.kit.filesystem' with several sub-packages and files, including 'Computer.java 59'. The main editor window displays the source code of 'Computer.java', which includes a 'private void printContent' method and a 'main' method. The 'Members' view on the right shows the class members, including 'import declarations', 'computerName : String', 'hdds : Vector<HDD>', 'Computer(String, HDD)', 'addDrive(HDD) : void', 'printContent() : void', 'printContent(Node, String) : void', and 'main(String[]) : void'.

```
32      printContent(f, "|-");
33    }
34  }
35
36
37  private void printContent(Node d, String ident) {
38    System.out.println("[" + ident + " " + d.getName());
39
40    ArrayList<Class? extends Node> list = new ArrayList<Class? extends Node>();
41    list.add(Directory.class);
42    list.add(ZipArchiv.class);
43    list.add(File.class);
44
45    if (d instanceof NodeContainer) {
46      NodeContainer e = (NodeContainer) d;
47      for (Class? extends Node T : list) {
48        ArrayList? extends Node tmp = e.get(T);
49        for (Node n : tmp) {
50          printContent(n, ident + "-");
51        }
52      }
53    }
```

- 1 Einleitung
- 2 Eclipse
- 3 Arrays
- 4 Random Style Guide
- 5 Getter/Setter
- 6 Konventionen
- 7 Kontrollstrukturen
- 8 Praxis
- 9 Abspann

```
1 public class Quiz {
2     public static void main(String[] args) {
3         String a = "";
4         String b;
5         if (a == b) {
6             System.out.println("Nyan Cat");
7         } else {
8             System.out.println("42");
9         }
10    }
11 }
```

- Was ist die Ausgabe?
- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?

## Ein Compiler-Fehler:

```
1 user@pc:~/Tutorium-03$ javac Quiz.java
2 Quiz.java:5: variable b might not have been initialized
3         if (a == b) {
4                 ^
5 1 error
```

- Installation (für Windows): [eclipse.org](http://eclipse.org)

- Window » Open Perspective » Java

- Window » Show Toolbar

- Window » Preferences » General » Editors » Text Editors

- Show line numbers
- Print margin column: 120

# Checkstyle: Installation

- Internetverbindung wird benötigt!
- 
- Work with:
- Klick auf
- Name: „Checkstyle“
- Warten
- Nun sollten zwei Einträge erscheinen
- „Checkstyle“ auswählen
- auf  klicken (und dann nochmal)
- „I accept the terms of the licence agreement“
- auf  klicken und dann herunterladen lassen
- „Warning: You are installing software [...]“ → klick auf
- Eclipse neustarten lassen (Klick auf )

- „Checkstyle.xml“ herunterladen: [tinyurl.com/checkstyle-ws](http://tinyurl.com/checkstyle-ws)

Bei jeden Java-Projekt wieder:

- **Project** » **Properties** » **Checkstyle**
- Check „checkstyle active for this project“
- Reiter **Local Check Configurations**
- **New...**
  - Type: „Internal Configuration“
  - Name: „KIT Checkstyle“
  - **Import** → „checkstyle.xml“ auswählen
  - **OK** klicken
- Reiter **Main** auswählen
- „KIT Checkstyle - (Local)“ auswählen
- **OK** klicken
- „The project needs to be rebuild [...]“ → **Yes**

Nochmal mit Screenshots: [martin-thoma.com/checkstyle](http://martin-thoma.com/checkstyle)

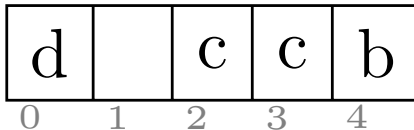


# Was sind Arrays ...

... und wozu braucht man sie?

- viele Werte in einem Variablennamen
- Elemente haben alle den selben Typ

⇒ zu jeden Typen gibt es Arrays



- Indices: 0, 1, 2, 3, 4
- Länge des Arrays: 5
- Erstes Element: d

Deklarieren:

```
int[] myArray; // Integer-Array
```

Deklarieren und instanziiieren:

```
int[] myArray = new int[3]; // Array mit 3 int-Werten
```

Deklarieren und initialisieren:

```
int[] myArray = {5, 3, 1}; // Definiere die Werte des Arrays  
// -> Größe ist implizit gegeben
```

(A) Geht, soll man aber nicht machen:

```
String myStringArray[];
```

(B) So ist es gut:

```
String[] myStringArray;
```

(A) Geht, soll man aber nicht machen:

```
String myStringArray[];
```

(B) So ist es gut:

```
String[] myStringArray;
```

Warum ist Variante (B) besser?

- Der Entwickler kann sofort den Typen sehen
- Konvention

(A) Geht, soll man aber nicht machen:

```
String myStringArray[];
```

(B) So ist es gut:

```
String[] myStringArray;
```

Warum ist Variante (B) besser?

- Der Entwickler kann sofort den Typen sehen

- Konvention

(A) Geht, soll man aber nicht machen:

```
String myStringArray[];
```

(B) So ist es gut:

```
String[] myStringArray;
```

Warum ist Variante (B) besser?

- Der Entwickler kann sofort den Typen sehen
- Konvention

- [JLS 7: Ab S. 291](#)
- [Java 7 API](#)
- [Java Tutorial](#)



# Antipattern: Yoda-Conditions



```
if (5 == count)
```

Using *if(constant == variable)* instead of *if(variable == constant)*, like *if(4 == foo)*. Because it's like saying "if blue is the sky" or "if tall is the man".

Source: [codinghorror.com](http://codinghorror.com)

Bitte nicht machen!

## ■ 1 Deklaration Zeile

Nicht so:

```
1 int level, size;
```

Sondern so:

```
1 int level; // indentation level
2 int size; // size of table
```

- Variablen immer dort initialisieren, wo sie deklariert werden  
Ausnahme: Initialisierungswert ist von vorherigen Berechnungen abhängig

# Antipattern: Stringly Typed

Used to describe an implementation that needlessly relies on strings.

Excessively stringly typed code is usually a pain to understand and detonates at runtime with errors that the compiler would normally find.

Source: [codinghorror.com](http://codinghorror.com)



Getter und Setter sind ...

- ... Methoden
- ... ein „Interface“
- ... **Zugriffsfunktionen** zur Abfrage und Änderung

Getter und Setter sind ...

- ... Methoden
- ... ein „Interface“
- ... **Zugriffsfunktionen** zur Abfrage und Änderung

Getter und Setter sind ...

- ... Methoden
- ... ein „Interface“
- ... [Zugriffsfunktionen](#) zur Abfrage und Änderung

Vorteile von Getter und Setter-Methoden sind ...

- ... (später auftretende) Nebenbedingungen beim get / set
- ... Validierung bei set
- ... Verbergen der Implementierung → Geheimnisprinzip

Vorteile von Getter und Setter-Methoden sind ...

- ... (später auftretende) Nebenbedingungen beim get / set
- ... Validierung bei set
- ... Verbergen der Implementierung → Geheimnisprinzip



Vorteile von Getter und Setter-Methoden sind ...

- ... (später auftretende) Nebenbedingungen beim get / set
- ... Validierung bei set
- ... Verbergen der Implementierung → Geheimnisprinzip

## Zugriffsmodifikatoren

Mit Hilfe von **Zugriffsmodifikatoren** (access modifiers) lassen sich die **Sichtbarkeiten** von Programmteilen regeln:

- **public** Element: Element ist für alle Klassen sichtbar
- **private** Element: Element ist nur innerhalb seiner Klasse sichtbar
- **protected** Element: Element ist nur innerhalb seiner Klasse, deren Subklassen und allen Klassen im selben Paket sichtbar → später mehr dazu
- **kein Modifier**: Element ist nur innerhalb seiner Klasse und der Klassen im selben Paket sichtbar → hier nicht so wichtig

## Zugriffsmodifikatoren

Mit Hilfe von **Zugriffsmodifikatoren** (access modifiers) lassen sich die **Sichtbarkeiten** von Programmteilen regeln:

- **public** Element: Element ist für alle Klassen sichtbar
- **private** Element: Element ist nur innerhalb seiner Klasse sichtbar
- **protected** Element: Element ist nur innerhalb seiner Klasse, deren Subklassen und allen Klassen im selben Paket sichtbar → später mehr dazu
- **kein Modifier**: Element ist nur innerhalb seiner Klasse und der Klassen im selben Paket sichtbar → hier nicht so wichtig

## Zugriffsmodifikatoren

Mit Hilfe von **Zugriffsmodifikatoren** (access modifiers) lassen sich die **Sichtbarkeiten** von Programmteilen regeln:

- **public** Element: Element ist für alle Klassen sichtbar
- **private** Element: Element ist nur innerhalb seiner Klasse sichtbar
- **protected** Element: Element ist nur innerhalb seiner Klasse, deren Subklassen und allen Klassen im selben Paket sichtbar → später mehr dazu
- **kein Modifier**: Element ist nur innerhalb seiner Klasse und der Klassen im selben Paket sichtbar → hier nicht so wichtig

## Zugriffsmodifikatoren

Mit Hilfe von **Zugriffsmodifikatoren** (access modifiers) lassen sich die **Sichtbarkeiten** von Programmteilen regeln:

- **public** Element: Element ist für alle Klassen sichtbar
- **private** Element: Element ist nur innerhalb seiner Klasse sichtbar
- **protected** Element: Element ist nur innerhalb seiner Klasse, deren Subklassen und allen Klassen im selben Paket sichtbar → später mehr dazu
- **kein Modifier**: Element ist nur innerhalb seiner Klasse und der Klassen im selben Paket sichtbar → hier nicht so wichtig

## Zugriffsmodifikatoren

Mit Hilfe von **Zugriffsmodifikatoren** (access modifiers) lassen sich die **Sichtbarkeiten** von Programmteilen regeln:

- **public** Element: Element ist für alle Klassen sichtbar
- **private** Element: Element ist nur innerhalb seiner Klasse sichtbar
- **protected** Element: Element ist nur innerhalb seiner Klasse, deren Subklassen und allen Klassen im selben Paket sichtbar → später mehr dazu
- **kein Modifier**: Element ist nur innerhalb seiner Klasse und der Klassen im selben Paket sichtbar → hier nicht so wichtig

Ab nun:

- Attribute sind (fast) immer private
- Methoden können auch private sein

## Student.java

```
1 public class Student {  
2     // die Attribute sind nun nach außen nicht mehr sichtbar  
3     private String name;  
4     private int semester;  
5     private int matriculationNumber;  
6  
7     public Student(String name, int semester, int matriculationNumber) {  
8         // hier wird wie gewohnt alles initialisiert  
9     }  
10 }
```

## Main.java

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Student maxMustermann = new Student("Max Mustermann", 3, 1234567);  
4         // hier bekommt man nun einen Compilerfehler  
5         maxMustermann.matriculationNumber = 3141592;  
6         // ...  
7     }  
8 }
```

## Neues Problem

Jetzt können wir Namen, Semester und Matrikelnummer von außen gar nicht mehr auslesen!



## Neues Problem

Jetzt können wir Namen, Semester und Matrikelnummer von außen gar nicht mehr auslesen!

Auch hierzu gibt es aber eine Lösung:

Mit **getter-Methoden** kann man den Lesezugriff auf Attribute wieder erlauben.

# Modifikatoren: Beispiel

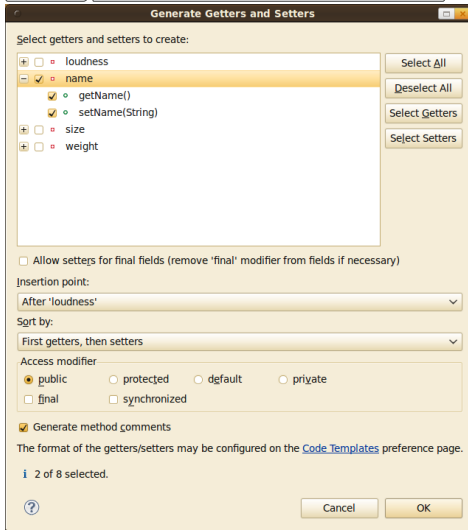
Student.java

```
1 public class Student {  
2     // ... Attribute, Konstruktor usw. ...  
3  
4  
5     // die getter-Methode für das Attribute 'name'  
6     public String getName() {  
7         return this.name;  
8     }  
9     // ... weitere getter-Methoden usw. ...  
10 }
```

Student.java

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Student maxMustermann = new Student("Max Mustermann", 3, 1234567);  
4         // liest den Namen und gibt ihn aus  
5         System.out.println(maxMustermann.getName());  
6         // ...  
7     }  
8 }
```

## Source » Generate Getters and Setters. . .



Typen:

- Implementierungskommentare:  
■ `/* blah */` und `// blah`
- Dokumentationskommentare: `/** blah */`

*Comments should not be enclosed in large boxes drawn with asterisks or other characters. Comments should never include special characters such as form-feed and backspace.*

Source: [Java Code Conventions](#), S. 7 - 9

Soll fast überall benutzt werden:

- Über jeder Klasse
- Über jedem Attribut
- Über jeder Methode (mit Annotations)

Es gibt folgende Annotations

- `@param` : Für die Parameter aller Methoden
- `@return` : Für den Rückgabewert vom Methoden
- `@author` : Nur für `class` und `interface` , erforderlich

Weitere Annotations:

- `@throws` : Angabe möglicher Fehlermeldungen

```
1 /**
2  * Sets the tool tip text.
3  *
4  * @param text the text of the tool tip
5  */
6 public void setToolTipText(String text) {
```

- Was ist hier schlecht?
- Wie könnte man es verbessern?

```
1 /**
2  * Registers the text to display in a tool tip.  The text
3  * displays when the cursor lingers over the component.
4  *
5  * @param text  the string to display.  If the text is null,
6  *              the tool tip is turned off for this component.
7  */
8 public void setToolTipText(String text) {
```



```
1 if (<Bedingung>) {  
2     // Anweisung für '<Bedingung> ist wahr'  
3 } else {  
4     // Anweisung für '<Bedingung> ist falsch'  
5 }
```

KEINE Schleife! → [if-schleife.de](http://if-schleife.de)

# if-Abfragen: else if

```
1 if (<Bedingung>) {  
2     // Anweisung für '<Bedingung> ist wahr'  
3 } else if (<andere Bedingung>) {  
4     // Anweisung für '<andere Bedingung> ist wahr'  
5 } else {  
6     // Anweisung für '<Bedingung> ist falsch'  
7 }
```

```
1 public class QuizIf {
2     public static void main(String[] a) {
3         int monat = 12;
4
5         if (monat == 12) {
6             System.out.println("12");
7         } else if (monat / 2 == 6) {
8             System.out.println("6");
9         } else {
10            System.out.println("2");
11        }
12    }
13 }
```

## ■ Syntax:

```
for ([INITIALISIERUNG; BEDINGUNG; UPDATE]) { ... }
```

```
1 for (int i = 0; i < 10; i++) {  
2     System.out.println(i);  
3 }
```

■ Syntax: `while ([BEDINGUNG]) { ... }`

```
1 while(true) {  
2     System.out.println("It's true!");  
3 }
```

- Syntax: `do { ... } while ([BEDINGUNG]);`
- Wo ist der Unterschied zu `while`?

```
1 while (bedingung) {  
2     // Anweisungen werden ausgeführt, solange bedingung == true  
3 }  
4  
5 do {  
6     // Anweisungen werden ausgeführt, solange bedingung == true  
7 } while (bedingung);
```

World.java

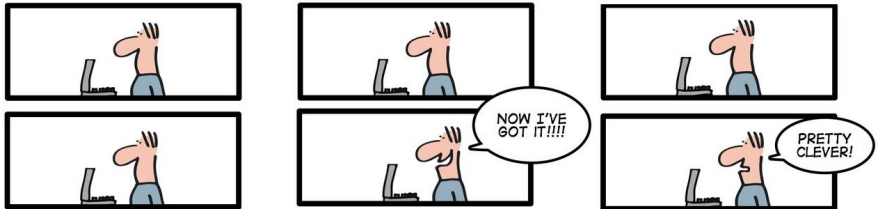
```
1 public class World {  
2     public static void main(String[] a) {  
3         int monat = 12;  
4         switch (monat) {  
5             case 1:  
6                 System.out.println("Es ist Januar!");  
7                 break;  
8             case 2:  
9                 System.out.println("Es ist Februar!");  
10                break;  
11            case 3:  
12                System.out.println("Es ist März!");  
13                break;  
14            default:  
15                // Anweisung, wenn monat keinen angegebenen Wert angenommen hat.  
16                System.out.println("Es ist Weihnachten!");  
17        }  
18    }  
19 }
```

Falls noch Zeit bleibt . . .



- 11. 05.11.2012
- 10. 12.11.2012
- 9. 19.11.2012
- 8. 26.11.2012
- 7. 03.12.2012
- 6. 10.12.2012
- 5. 17.12.2012: Video „Library“ zeigen
  - 24.12.2012: Heiligabend - **Kein Tutorium**
  - 31.12.2012: Silvester - Kein Tutorium
- 4. 07.01.2013
- 3. 14.01.2013
- 2. 21.01.2013
- 1. 28.01.2013
- 0. 04.02.2013
  - 10.02.2013: Ende der Vorlesungszeit des WS 2012/2013 (**Quelle**)

# Vielen Dank für eure Aufmerksamkeit!



Geek and Poke: Coders Great Moments

*ISN'T IT A GREAT  
FEELING WHEN YOU  
FINALLY GET YOUR OWN  
CODE, YOU'VE WRITTEN  
MONTHS BEFORE?*