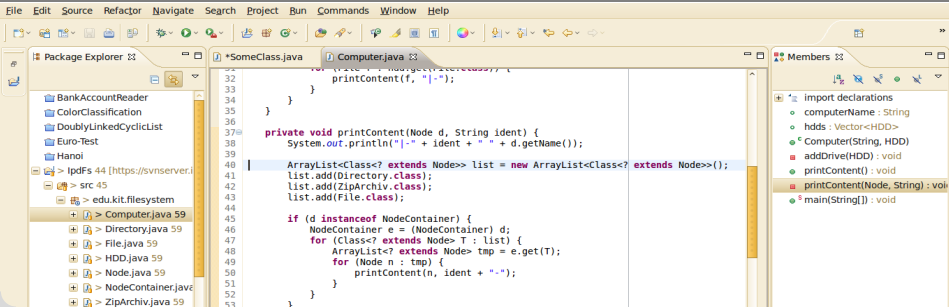


# Programmieren-Tutorium Nr. 10 bei Martin Thoma

Javadoc, Bytecode,

Martin Thoma | 12. November 2012

FAKULTÄT FÜR INFORMATIK



The screenshot shows an IDE with the following components:

- Package Explorer (Left):** Displays a project structure with folders like 'BankAccountReader', 'ColorClassification', 'DoublyLinkedCyclicList', 'Euro-Test', 'Hanoi', and 'src 45'. Under 'src 45', there is a folder 'edu.kit.filesystem' containing several Java files, including 'Computer.java 59' which is currently selected.
- Editor (Center):** Shows the source code of 'Computer.java'. The code includes a package declaration, imports, and a class 'Computer' that extends 'Node'. It features a 'printContent' method that uses 'System.out.println' and an 'ArrayList' to store and process nodes. The line 'ArrayList<Class? extends Node> list = new ArrayList<Class? extends Node>();' is highlighted.
- Members (Right):** Displays a list of members for the selected class, including 'import declarations', 'computerName : String', 'hdds : Vector<HDD>', 'Computer(String, HDD)', 'addDrive(HDD) : void', 'printContent() : void', 'printContent(Node, String) : void', and 'main(String[]) : void'.

- 1 Einleitung
- 2 Nachbesprechung: 1. ÜB
- 3 Exkurs
- 4 Dies und Das
- 5 Hinweise zum ÜB 2
- 6 Abspann

## Quiz.java

```
1 public class QuizIf {  
2     public static void main(String[] a) {  
3         float a = 0.1;  
4         float b = 0.1;  
5         if (0.01 == a * b) {  
6             System.out.println("Alpha");  
7         } else {  
8             System.out.println("Beta");  
9         }  
10    }  
11 }
```

- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?
- Gibt es eine Ausgabe? Welche?

## Quiz.java

```
1 public class QuizIf {  
2     public static void main(String[] a) {  
3         float a = 0.1f;  
4         float b = 0.1f;  
5         if (0.01 == a * b) {  
6             System.out.println("Alpha");  
7         } else {  
8             System.out.println("Beta");  
9         }  
10    }  
11 }
```

- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?
- Gibt es eine Ausgabe? Welche?

## Quiz.java

```
1 public class QuizIf {  
2     public static void main(String[] args) {  
3         float a = 0.1f;  
4         float b = 0.1f;  
5         if (0.01 == a * b) {  
6             System.out.println("Alpha");  
7         } else {  
8             System.out.println("Beta");  
9         }  
10    }  
11 }
```

- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?
- Gibt es eine Ausgabe? Welche?

Welche Vorteile bieten Ganzzahl-Variablen im Vergleich zu Gleitkomma-Variablen?

- **Speicherplatz?** Nein, vgl. `long` und `float`
- **Geschwindigkeit?** Kommt drauf an: Wenn keine weitere Umrechnung nötig ist und die Gleitkommazahl nicht kleiner ist, eher ja.
- **Genauigkeit?** Ja.  
Beispiel:  $(0,1)_{10} = (0,00011)_2$   
vgl. Java-Puzzle

Welche Vorteile bieten Ganzzahl-Variablen im Vergleich zu Gleitkomma-Variablen?

- **Speicherplatz?** Nein, vgl. `long` und `float`
- **Geschwindigkeit?** Kommt drauf an: Wenn keine weitere Umrechnung nötig ist und die Gleitkommazahl nicht kleiner ist, eher ja.
- **Genauigkeit?** Ja.  
Beispiel:  $(0,1)_{10} = (0,00011)_2$   
vgl. Java-Puzzle

Welche Vorteile bieten Ganzzahl-Variablen im Vergleich zu Gleitkomma-Variablen?

- **Speicherplatz?** Nein, vgl. `long` und `float`
- **Geschwindigkeit?** Kommt drauf an: Wenn keine weitere Umrechnung nötig ist und die Gleitkommazahl nicht kleiner ist, eher ja.
- **Genauigkeit?** Ja.  
Beispiel:  $(0,1)_{10} = (0,0\overline{0011})_2$   
vgl. Java-Puzzle



Welche Vorteile bieten Ganzzahl-Variablen im Vergleich zu Gleitkomma-Variablen?

- **Speicherplatz?** Nein, vgl. `long` und `float`
- **Geschwindigkeit?** Kommt drauf an: Wenn keine weitere Umrechnung nötig ist und die Gleitkommazahl nicht kleiner ist, eher ja.
- **Genauigkeit?** Ja.  
Beispiel:  $(0,1)_{10} = (0,0\overline{0011})_2$   
vgl. Java-Puzzle

Welche Vorteile bieten Ganzzahl-Variablen im Vergleich zu Gleitkomma-Variablen?

- **Speicherplatz?** Nein, vgl. `long` und `float`
- **Geschwindigkeit?** Kommt drauf an: Wenn keine weitere Umrechnung nötig ist und die Gleitkommazahl nicht kleiner ist, eher ja.
- **Genauigkeit?** Ja.

Beispiel:  $(0,1)_{10} = (0,0\overline{0011})_2$

vgl. Java-Puzzle

Welche Vorteile bieten Ganzzahl-Variablen im Vergleich zu Gleitkomma-Variablen?

- **Speicherplatz?** Nein, vgl. `long` und `float`
- **Geschwindigkeit?** Kommt drauf an: Wenn keine weitere Umrechnung nötig ist und die Gleitkommazahl nicht kleiner ist, eher ja.
- **Genauigkeit?** Ja.  
Beispiel:  $(0,1)_{10} = (0,0\overline{0011})_2$   
vgl. Java-Puzzle

## Regel

Kommentiert, was ihr macht.

Nicht wie ihr es macht.

- Gut zu kommentieren ist schwer
- Viel (fremden) Code ansehen hilft
- Eigenen Code nach Jahren ansehen hilft
- JEDER Kommentar ist für Java-Entwickler gedacht

⇒ Kommentare à la „Methode“ oder „Methodensignatur“ sind nutzlos!

- Javadoc-Kommentare sind für Entwickler, die nichts von eurem Code sehen können, ihn aber dennoch nutzen wollen

## Regel

Kommentiert, was ihr macht.

Nicht wie ihr es macht.

- Gut zu kommentieren ist schwer
- Viel (fremden) Code ansehen hilft
- Eigenen Code nach Jahren ansehen hilft
- JEDER Kommentar ist für Java-Entwickler gedacht

⇒ Kommentare à la „Methode“ oder „Methodensignatur“ sind nutzlos!

- Javadoc-Kommentare sind für Entwickler, die nichts von eurem Code sehen können, ihn aber dennoch nutzen wollen

## Regel

Kommentiert, was ihr macht.

Nicht wie ihr es macht.

- Gut zu kommentieren ist schwer
- Viel (fremden) Code ansehen hilft
- Eigenen Code nach Jahren ansehen hilft
- JEDER Kommentar ist für Java-Entwickler gedacht

⇒ Kommentare à la „Methode“ oder „Methodensignatur“ sind nutzlos!

- Javadoc-Kommentare sind für Entwickler, die nichts von eurem Code sehen können, ihn aber dennoch nutzen wollen

## Regel

Kommentiert, was ihr macht.

Nicht wie ihr es macht.

- Gut zu kommentieren ist schwer
- Viel (fremden) Code ansehen hilft
- Eigenen Code nach Jahren ansehen hilft
- JEDER Kommentar ist für Java-Entwickler gedacht

⇒ Kommentare à la „Methode“ oder „Methodensignatur“ sind nutzlos!

- Javadoc-Kommentare sind für Entwickler, die nichts von eurem Code sehen können, ihn aber dennoch nutzen wollen

## Regel

Kommentiert, was ihr macht.

Nicht wie ihr es macht.

- Gut zu kommentieren ist schwer
  - Viel (fremden) Code ansehen hilft
  - Eigenen Code nach Jahren ansehen hilft
  - JEDER Kommentar ist für Java-Entwickler gedacht
- ⇒ Kommentare à la „Methode“ oder „Methodensignatur“ sind nutzlos!
- Javadoc-Kommentare sind für Entwickler, die nichts von eurem Code sehen können, ihn aber dennoch nutzen wollen



## Regel

Kommentiert, was ihr macht.

Nicht wie ihr es macht.

- Gut zu kommentieren ist schwer
  - Viel (fremden) Code ansehen hilft
  - Eigenen Code nach Jahren ansehen hilft
  - JEDER Kommentar ist für Java-Entwickler gedacht
- ⇒ Kommentare à la „Methode“ oder „Methodensignatur“ sind nutzlos!
- Javadoc-Kommentare sind für Entwickler, die nichts von eurem Code sehen können, ihn aber dennoch nutzen wollen

```
1 public boolean isStreetLegal() {
2     if(bell==true && light==true) { //Wenn das Fahrrad eine Klingel und ein Licht hat...
3         legal = true; //dann ist es auf der Straße zugelassen
4     }
5     else { //wenn es keine Beleuchtung und/oder Licht hat, ...
6         legal = false; // dann ist es nicht zugelassen
7     }
8     return legal; //Zurückgabe des Attributs ob es zugelassen ist
9 }
10
11 public int getPriceFull(){ //Methode zur Berechnung des Gesamtpreises des Fahrrades
12     int priceFull = shift.getPrice() + price + wheels.getPrice(); /*
13     Berechnung des Preises durch Addition der Einzelpreise*/
14     return priceFull; //Zurückgabe des Gesamtpreises
15 }
```

```
1 public boolean isStreetLegal() {
2     if(bell==true && light==true) { //Wenn das Fahrrad eine Klingel und ein Licht hat...
3         legal = true; //dann ist es auf der Straße zugelassen
4     }
5     else { //wenn es keine Beleuchtung und/oder Licht hat, ...
6         legal = false; // dann ist es nicht zugelassen
7     }
8     return legal; //Zurückgabe des Attributs ob es zugelassen ist
9 }
10
11 public int getPriceFull(){ //Methode zur Berechnung des Gesamtpreises des Fahrrades
12     int priceFull = shift.getPrice() + price + wheels.getPrice(); /*
13     Berechnung des Preises durch Addition der Einzelpreise*/
14     return priceFull; //Zurückgabe des Gesamtpreises
15 }
```

- An sich gute Kommentare
- Wäre besser als Javadoc direkt über der Methode

```
1 //Konstruktor
2 /** erzeugt ein neues Objekt und initialisiert die Attribute */
3 Gears(byte chainwheel0, byte rearsprocket0, int price0) {
4     chainwheel = chainwheel0;
5     rearsprocket = rearsprocket0;
6     price = price0;
7 }
8
9 // Methode
10 /** gibt die Anzahl der Gänge zurück */
11 short getNumberOfGears() {
12     short numbergears;
13
14     numbergears = (short) (rearsprocket * chainwheel);
15     return numbergears;
16 }
```

- Zeile 1 & 2 bieten einem Entwickler nicht mehr Informationen ⇒ nutzlos
- Sonst okay
- Bitte nicht `chainwheel0`, sondern `chainwheel` und später `this`-Operator nutzen  
Gibt in Zukunft -0,5 Punkte
- Gute Zeilenlänge 😊

# Kommentare: Beispiel 3

```
1  /**
2   * Methode, die ein neues Stadtrad erstellt.
3   * @return neues Stadtrad
4   */
5  public Bike createCityBike() {           //Methodensignatur der Methode createCityBike
6      Wheels cityWheels = new Wheels(559,50f,10000);           //Räder des Stadtrads erstellen
7      Gears cityGears = new Gears(3,1,5000);                 //Gangschaltung des Stadtrads erstellen
8      Bike newCityBike = new Bike(cityGears, cityWheels, "Stahl", "CB105", true, true, 30000); //Stadrad erstellen
9      return newCityBike;           //Stadtrad zurückgeben
10 }
```

# Kommentare: Beispiel 3

```
1  /**
2   * Methode, die ein neues Stadtrad erstellt.
3   * @return neues Stadtrad
4   */
5  public Bike createCityBike() {           //Methodensignatur der Methode createCityBike
6      Wheels cityWheels = new Wheels(559,50f,10000);           //Räder des Stadtrads erstellen
7      Gears cityGears = new Gears(3,1,5000);           //Gangschaltung des Stadtrads erstellen
8      Bike newCityBike = new Bike(cityGears, cityWheels, "Stahl", "CB105", true, true, 30000); //Stadtrad erstellen
9      return newCityBike;           //Stadtrad zurückgeben
10 }
```

- Javadoc ist okay
- „Methodensignatur“-Kommentar in Z. 5 ist nutzlos
- Kommentare in Z. 7 - 9 sind nutzlos
- Z. 8 ist arg lang → den Kommentar hätte man einfach über die Zeile schreiben können.

## Regel

Der Präfix „is“ sollte für boolesche Variablen und Methoden mit dem Rückgabewert `boolean` genutzt werden.

## Beispiele

`isSet`, `isVisible`, `isFinished`, `isFound`, `isOpen`

Auch okay sind „has“, „should“ oder ähnliche Prefixe.

## Beispiele

```
boolean hasLicense();  
boolean canEvaluate();  
boolean shouldAbort = false;
```



## Regel

Der Präfix „is“ sollte für boolesche Variablen und Methoden mit dem Rückgabewert `boolean` genutzt werden.

## Beispiele

`isSet`, `isVisible`, `isFinished`, `isFound`, `isOpen`

Auch okay sind „has“, „should“ oder ähnliche Prefixe.

## Beispiele

```
boolean hasLicense();  
boolean canEvaluate();  
boolean shouldAbort = false;
```

## Regel

Der Präfix „is“ sollte für boolesche Variablen und Methoden mit dem Rückgabewert `boolean` genutzt werden.

## Beispiele

`isSet`, `isVisible`, `isFinished`, `isFound`, `isOpen`

Auch okay sind „has“, „should“ oder ähnliche Prefixe.

## Beispiele

```
boolean hasLicense();  
boolean canEvaluate();  
boolean shouldAbort = false;
```

## Regel

Der Präfix „is“ sollte für boolesche Variablen und Methoden mit dem Rückgabewert `boolean` genutzt werden.

## Beispiele

`isSet`, `isVisible`, `isFinished`, `isFound`, `isOpen`

Auch okay sind „has“, „should“ oder ähnliche Prefixe.

## Beispiele

```
boolean hasLicense();  
boolean canEvaluate();  
boolean shouldAbort = false;
```

## Negativbeispiel: So nicht!

```
boolean bell;  
boolean light;
```

## Positivbeispiel: Aber so

```
boolean hasBell;  
boolean hasLight;
```

In Zukunft: -0,5 Punkte

## Negativbeispiel: So nicht!

```
boolean bell;  
boolean light;
```

## Positivbeispiel: Aber so

```
boolean hasBell;  
boolean hasLight;
```

In Zukunft: -0,5 Punkte

## Negativbeispiel: So nicht!

```
boolean bell;  
boolean light;
```

## Positivbeispiel: Aber so

```
boolean hasBell;  
boolean hasLight;
```

In Zukunft: -0,5 Punkte

# Boolean: Was ist mit Gettern/Settern?

```
public class TestBoolean {  
    private boolean isActive;  
  
    public boolean isActive() {  
        return isActive;  
    }  
}
```

## Hinweis

Es ist okay, wenn ein Attribut genauso heißt wie eine Methode

Wenn man 3 Gänge vorne und 7 hinten hat, wie viele Gänge gibt es?

**Antwort:**  $3 \cdot 7 = 21$

**Erklärung:** Sei  $\{a, b, c\}$  die Menge der vorderen Gänge und  $\{1, 2, 3, 4, 5, 6, 7\}$  die Menge der hinteren Gänge.

Dann gibt es folgende Kombinationen:

a1, a2, a3, a4, a5, a6, a7

b1, b2, b3, b4, b5, b6, b7

c1, c2, c3, c4, c5, c6, c7



Wenn man 3 Gänge vorne und 7 hinten hat, wie viele Gänge gibt es?

**Antwort:**  $3 \cdot 7 = 21$

**Erklärung:** Sei  $\{a, b, c\}$  die Menge der vorderen Gänge und  $\{1, 2, 3, 4, 5, 6, 7\}$  die Menge der hinteren Gänge.

Dann gibt es folgende Kombinationen:

a1, a2, a3, a4, a5, a6, a7

b1, b2, b3, b4, b5, b6, b7

c1, c2, c3, c4, c5, c6, c7

Wenn man 3 Gänge vorne und 7 hinten hat, wie viele Gänge gibt es?

**Antwort:**  $3 \cdot 7 = 21$

**Erklärung:** Sei  $\{a, b, c\}$  die Menge der vorderen Gänge und  $\{1, 2, 3, 4, 5, 6, 7\}$  die Menge der hinteren Gänge.

Dann gibt es folgende Kombinationen:

a1, a2, a3, a4, a5, a6, a7

b1, b2, b3, b4, b5, b6, b7

c1, c2, c3, c4, c5, c6, c7

Wenn man 3 Gänge vorne und 7 hinten hat, wie viele Gänge gibt es?

**Antwort:**  $3 \cdot 7 = 21$

**Erklärung:** Sei  $\{a, b, c\}$  die Menge der vorderen Gänge und  $\{1, 2, 3, 4, 5, 6, 7\}$  die Menge der hinteren Gänge.

Dann gibt es folgende Kombinationen:

a1, a2, a3, a4, a5, a6, a7

b1, b2, b3, b4, b5, b6, b7

c1, c2, c3, c4, c5, c6, c7

Mit Eclipse:

- Alles markieren: `ctrl` + `A`
- Formatieren: `ctrl` + `↑` + `F`

Falsche Formatierung gibt in Zukunft pro Fehler -0,5 Punkte.  
Auch Folgefehler geben Punktabzug!

# Aussagekräftige Variablen!

In Zukunft: -1 P. bis -5 P. für Variablennamen wie „Kr“ für Kettenräder oder „Pr“ für Preis!

```
_____ Baby.java _____  
1 public class Baby {  
2     public String name;  
3     public static int size;  
4  
5     public Baby(String name) {  
6         this.name = name;  
7         size = 46;  
8     }  
9 }
```

```
_____ World.java _____  
1 public class World {  
2     public static void main(String[] args) {  
3         Baby alice = new Baby("Alice");  
4         alice.size = 42;  
5  
6         Baby bob = new Baby("Bob");  
7         bob.size = 56;  
8  
9         System.out.println("Alice: " + alice.size);  
10        System.out.println("Bob: " + bob.size);  
11    }  
12 }
```

- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?
- Gibt es eine Ausgabe? Welche?

# Quiz: Antwort

Ausgabe:

- Alice: 56
- Bob: 56

Warum?

- `static` macht ein Attribut zu einem „**Klassenattribut**“
- Das Attribut gehört dann nicht mehr den einzelnen Objekten
- Und sollte auch nicht über Objecte aufgerufen werden!
- Schlecht: `alice.size;`
- Auch schlecht: `alice.getSize();`
- Besser: `Baby.size;` oder `Baby.getSize();`
- In Zukunft: min. -2 P. für falsche `static`-Verwendung

# Quiz: Antwort

Ausgabe:

- Alice: 56
- Bob: 56

Warum?

- `static` macht ein Attribut zu einem „**Klassenattribut**“
- Das Attribut gehört dann nicht mehr den einzelnen Objekten
- Und sollte auch nicht über Objecte aufgerufen werden!
- Schlecht: `alice.size;`
- Auch schlecht: `alice.getSize();`
- Besser: `Baby.size;` oder `Baby.getSize();`
- In Zukunft: min. -2 P. für falsche `static`-Verwendung



# Quiz: Antwort

Ausgabe:

- Alice: 56
- Bob: 56

Warum?

- `static` macht ein Attribut zu einem „**Klassenattribut**“
- Das Attribut gehört dann nicht mehr den einzelnen Objekten
- Und sollte auch nicht über Objecte aufgerufen werden!
- Schlecht: `alice.size;`
- Auch schlecht: `alice.getSize();`
- Besser: `Baby.size;` oder `Baby.getSize();`
- In Zukunft: min. -2 P. für falsche `static`-Verwendung

# Quiz: Antwort

Ausgabe:

- Alice: 56
- Bob: 56

Warum?

- `static` macht ein Attribut zu einem „**Klassenattribut**“
- Das Attribut gehört dann nicht mehr den einzelnen Objekten
- Und sollte auch nicht über Objecte aufgerufen werden!
- Schlecht: `alice.size;`
- Auch schlecht: `alice.getSize();`
- Besser: `Baby.size;` oder `Baby.getSize();`
- In Zukunft: min. -2 P. für falsche `static`-Verwendung

# Quiz: Antwort

Ausgabe:

- Alice: 56
- Bob: 56

Warum?

- `static` macht ein Attribut zu einem „**Klassenattribut**“
- Das Attribut gehört dann nicht mehr den einzelnen Objekten
- Und sollte auch nicht über Objecte aufgerufen werden!
- Schlecht: `alice.size;`
- Auch schlecht: `alice.getSize();`
- Besser: `Baby.size;` oder `Baby.getSize();`
- In Zukunft: min. -2 P. für falsche `static`-Verwendung

# Quiz: Antwort

Ausgabe:

- Alice: 56
- Bob: 56

Warum?

- `static` macht ein Attribut zu einem „**Klassenattribut**“
- Das Attribut gehört dann nicht mehr den einzelnen Objekten
- Und sollte auch nicht über Objecte aufgerufen werden!
- Schlecht: `alice.size;`
- Auch schlecht: `alice.getSize();`
- Besser: `Baby.size;` oder `Baby.getSize();`
- In Zukunft: min. -2 P. für falsche `static`-Verwendung

# Quiz: Antwort

Ausgabe:

- Alice: 56
- Bob: 56

Warum?

- `static` macht ein Attribut zu einem „**Klassenattribut**“
- Das Attribut gehört dann nicht mehr den einzelnen Objekten
- Und sollte auch nicht über Objecte aufgerufen werden!
- Schlecht: `alice.size;`
- Auch schlecht: `alice.getSize();`
- Besser: `Baby.size;` oder `Baby.getSize();`
- In Zukunft: min. -2 P. für falsche `static`-Verwendung

## Hinweis

Das folgende ist nicht Prüfungsrelevant!  
Also zurücklehnen und genießen :-)

Mit dem Befehl

```
javap -c SimpleLoop
```

könnt ihr euch den Java-Bytecode ansehen.

```
_____ SimpleLoop.java _____  
1 public class SimpleLoop {  
2     public static void main(String[] args) {  
3         for (int i = -5; i < 15; ++i) {  
4             System.out.println(i + ": " + i * i);  
5         }  
6     }  
7 }  
8 }
```

# Bytecode von SimpleLoop.java

Compiled from "SimpleLoop.java"

```
public class SimpleLoop extends java.lang.Object{  
    public SimpleLoop();
```

Code:

```
0:    aload_0  
1:    invokespecial    #1; //Method java/lang/Object."<init>":()V  
4:    return
```

```
public static void main(java.lang.String[]);
```

Code:

```
0:    bipush    -5  
2:    istore_1  
3:    iload_1  
4:    bipush    15  
6:    if_icmpge    46  
9:    getstatic    #2; //Field java/lang/System.out:Ljava/io/PrintStream;  
12:    new        #3; //class java/lang/StringBuilder  
15:    dup  
16:    invokespecial    #4; //Method java/lang/StringBuilder."<init>":()V  
19:    iload_1  
20:    invokevirtual    #5; //Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;  
23:    ldc        #6; //String :  
25:    invokevirtual    #7; //Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/String;  
28:    iload_1  
29:    iload_1  
30:    imul  
31:    invokevirtual    #5; //Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
```



# Bytecode von SimpleLoop.java: Anfang

Compiled from "SimpleLoop.java"

```
public class SimpleLoop extends java.lang.Object{  
    public SimpleLoop();
```

Code:

```
0:    aload_0  
1:    invokespecial    #1; //Method java/lang/Object."<init>":()V  
4:    return
```

- **aload\_0**: Lade eine Objektreferenz aus dem Array der lokalen Variablen auf den Operandenstapel. ([Quelle](#))
- **iload\_1**: Lade den int-Wert einer lokalen Variablen auf den Operandenstapel. ([Quelle](#))
- **invokespecial [method-spec]**: invoke method belonging to a specific class ([Quelle](#))

# Bytecode von SimpleLoop.java: Ende

```
public static void main(java.lang.String[]);
```

Code:

```
0:  bipush    -5
2:  istore_1
3:  iload_1
4:  bipush    15
6:  if_icmpge 46
9:  getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;
12: new       #3; //class java/lang/StringBuilder
15:  dup
16:  invokespecial #4; //Method java/lang/StringBuilder."<init>":()V
19:  iload_1
20:  invokevirtual #5; //Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
23:  ldc       #6; //String :
25:  invokevirtual #7; //Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/String;
28:  iload_1
29:  iload_1
30:  imul
31:  invokevirtual #5; //Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
34:  invokevirtual #8; //Method java/lang/StringBuilder.toString:()Ljava/lang/String;
37:  invokevirtual #9; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
40:  iinc      1, 1
43:  goto      3
46:  return
```

# Interessanter Teil des Bytecodes

Compiled from "SimpleLoop.java"

```
public class SimpleLoop extends java.lang.Object{  
    public SimpleLoop();
```

Code:

```
0:    aload_0  
1:    invokespecial    # 1; //Method java/lang/Object."<init>":()V  
4:    return
```

```
public static void main(java.lang.String[]);
```

Code:

```
0:    bipush    -5  
2:    istore_1    /* Speichere einen int-Wert in das Array der lokalen Variablen */  
3:    iload_1    /* Lade den int-Wert einer lokalen Variablen auf den Operandenstapel */  
4:    bipush    15 /* lege 15 auf den Operandenstapel */  
6:    if_icmpge 46 /* if_icmpge pops the top two ints off the stack  
and compares them. If value2 is greater than or equal to value1,  
execution branches to the address (pc + branchoffset), where pc  
is the address of the if_icmpge opcode in the bytecode and branchoffset  
is a 16-bit signed integer parameter following the if_icmpge opcode in  
the bytecode. If value2 is less than value1, execution continues at the  
next instruction.*/  
9-37: /* String erstellen, i*i berechnen, String ausgeben */  
40:    iinc      1, 1 /* iinc <varnum> <n> increments the int held in the local variable <varnum> by <n> */  
43:    goto      3  
46:    return  
}
```

## Java™ Platform Standard Ed. 6

[All Classes](#)

Packages

[java.applet](#)

[java.awt](#)

[java.awt.color](#)

### All Classes

[AbstractAction](#)

[AbstractAnnotationValueVisitor6](#)

[AbstractBorder](#)

[AbstractButton](#)

[AbstractCellEditor](#)

[AbstractCollection](#)

[AbstractColorChooserPanel](#)

[AbstractDocument](#)

[AbstractDocument.AttributeContext](#)

[AbstractDocument.Content](#)

[AbstractDocument.ElementEdit](#)

[AbstractElementVisitor6](#)

[AbstractExecutorService](#)

[AbstractInterruptibleChannel](#)

[AbstractLayoutCache](#)

[AbstractLayoutCache.NodeDimensions](#)

[AbstractList](#)

[AbstractListModel](#)

[AbstractMap](#)

[AbstractMap.SimpleEntry](#)

[AbstractMap.SimpleImmutableEntry](#)

[AbstractMarshallerImpl](#)

[AbstractMethodError](#)

[AbstractOwnableSynchronizer](#)

[AbstractPackage](#)

## Overview Package Class Use Tree Deprecated Index Help

PREV NEXT

[FRAMES](#) [NO FRAMES](#)

## Java™ Platform, Standard Edition 6 API Specification

This document is the API specification for version 6 of the Java™ Platform, Standard Edition 6.

See:

[Description](#)

### Packages

<a href="#">java.applet</a>	Provides the classes necessary for applets to run in a context.
<a href="#">java.awt</a>	Contains all of the classes for the Abstract Window Toolkit (AWT).
<a href="#">java.awt.color</a>	Provides classes for color space and color management.
<a href="#">java.awt.datatransfer</a>	Provides interfaces and classes for data transfer between applications.
<a href="#">java.awt.dnd</a>	Drag and Drop is a direct manipulation mechanism to transfer information between GUIs.
<a href="#">java.awt.event</a>	Provides interfaces and classes for event handling.
<a href="#">java.awt.font</a>	Provides classes and interfaces for font rendering.
<a href="#">java.awt.geom</a>	Provides the Java 2D classes for geometry.
<a href="#">java.awt.image</a>	Provides classes and interfaces for image processing.

Java™ Platform  
Standard Ed. 7

All Classes

Packages

java.applet  
java.awt  
java.awt.color  
java.awt.datatransfer

All Classes

AbstractAction  
AbstractAnnotationValueVisitor6  
AbstractAnnotationValueVisitor7  
AbstractBorder  
AbstractButton  
AbstractCellEditor  
AbstractCollection  
AbstractColorChooserPanel  
AbstractDocument  
AbstractDocument.AttributeContext  
AbstractDocument.Content  
AbstractDocument.ElementEdit  
AbstractElementVisitor6  
AbstractElementVisitor7  
AbstractExecutorService  
AbstractInterruptibleChannel  
AbstractLayoutCache  
AbstractLayoutCache.NodeDimensions  
AbstractList  
AbstractListModel  
AbstractMap  
AbstractMap.SimpleEntry  
AbstractMap.SimpleImmutableEntry  
AbstractMarshallerImpl  
AbstractMethodError  
AbstractObservable

Overview Package Class Use Tree Deprecated Index Help

Prev Next Frames No Frames

## Java™ Platform, Standard Edition 7 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet
java.awt	Contains all of the classes for creating user interface
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data
java.awt.dnd	Drag and Drop is a direct manipulation gesture for information between two entities logically associated
java.awt.event	Provides interfaces and classes for dealing with events
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing
java.awt.im	Provides classes and interfaces for the input method
java.awt.im.spi	Provides interfaces that enable the development of
java.awt.image	Provides classes for creating and modifying images
java.awt.image.renderable	Provides classes and interfaces for producing renderable
java.awt.print	Provides classes and interfaces for a general printing
java.beans	Contains classes related to developing beans -- components

- Order erstellen, in dem die Javadoc landen soll
- In den Ordner mit euren Quelldateien wechseln
- Befehl `javadoc -d ../pfad/zum/javadoc/ordner/ *`

# Javadoc erzeugen

```
1 moose@pc07:~/Downloads/prog-ws1213/Blatt-01$ ls
2 README.md student-solution
3 moose@pc07:~/Downloads/prog-ws1213/Blatt-01$ mkdir javadoc
4 moose@pc07:~/Downloads/prog-ws1213/Blatt-01$ ls
5 javadoc README.md student-solution
6 moose@pc07:~/Downloads/prog-ws1213/Blatt-01$ cd student-solution/
7 moose@pc07:~/Downloads/prog-ws1213/Blatt-01/student-solution$ ls
8 BikeFactory.java Bike.java doc Frame.java Gears.java test Wheels.java
9 moose@pc07:~/Downloads/prog-ws1213/Blatt-01/student-solution$ javadoc -d ../javadoc/ *
10 Loading source file BikeFactory.java...
11 Loading source file Bike.java...
12 Loading source file Frame.java...
13 Loading source file Gears.java...
14 Loading source file Wheels.java...
15 Wheels.java:4: warning: unmappable character for encoding UTF8
16 * Der Felgendurchmesser und die ReifenstXrke modellieren
17 ~
18 Wheels.java:5: warning: unmappable character for encoding UTF8
19 * die RXder eines Fahrrads.
20 ~
21 Wheels.java:6: warning: unmappable character for encoding UTF8
22 * Der Felgendurchmesser betrXgt maximal 700mm
23 ~
24 Wheels.java:7: warning: unmappable character for encoding UTF8
25 * und die ReifenstXrke betrXgt maximal 60mm.
26 ~
27 Wheels.java:7: warning: unmappable character for encoding UTF8
28 * und die ReifenstXrke betrXgt maximal 60mm.
29 ~
30 Wheels.java:18: warning: unmappable character for encoding UTF8
```

# Javadoc erzeugen


```
31      * Konstruktor fXr "Wheels".
32      ^
33 Wheels.java:44: warning: unmappable character for encoding UTF8
34       + "als Wert ungXltig. Maximaler Feldendurschmeeser " +
35       ^
36 Wheels.java:45: warning: unmappable character for encoding UTF8
37       "betrXgt 700mm. Bitte Wert Xndern.");
38       ^
39 Wheels.java:45: warning: unmappable character for encoding UTF8
40       "betrXgt 700mm. Bitte Wert Xndern.");
41       ^
42 Wheels.java:58: warning: unmappable character for encoding UTF8
43       + " als Wert ungXltig. Maximalee ReifenstXrke " +
44       ^
45 Wheels.java:58: warning: unmappable character for encoding UTF8
46       + " als Wert ungXltig. Maximalee ReifenstXrke " +
47       ^
48 Wheels.java:59: warning: unmappable character for encoding UTF8
49       "betrXgt 60mm. Bitte Wert Xndern.");
50       ^
51 Wheels.java:59: warning: unmappable character for encoding UTF8
52       "betrXgt 60mm. Bitte Wert Xndern.");
53       ^
54 Loading source files for package doc...
55 javadoc: warning - No source files for package doc
56 Loading source files for package test...
57 Constructing Javadoc information...
58 javadoc: warning - No source files for package doc
59 javadoc: warning - No source files for package test
```




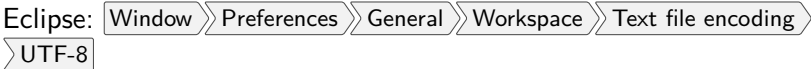
# Javadoc erzeugen

```
60 Standard Doclet version 1.6.0_24
61 Building tree for all the packages and classes...
62 Generating ../javadoc/Velo/Bike.html...
63 Generating ../javadoc/Velo/BikeFactory.html...
64 Generating ../javadoc/Velo/Frame.html...
65 Generating ../javadoc/Velo/Gears.html...
66 Generating ../javadoc/Velo/Wheels.html...
67 Generating ../javadoc/Velo/package-frame.html...
68 Generating ../javadoc/Velo/package-summary.html...
69 Generating ../javadoc/Velo/package-tree.html...
70 Generating ../javadoc/constant-values.html...
71 Building index for all the packages and classes...
72 Generating ../javadoc/overview-tree.html...
73 Generating ../javadoc/index-all.html...
74 Generating ../javadoc/deprecated-list.html...
75 Building index for all classes...
76 Generating ../javadoc/allclasses-frame.html...
77 Generating ../javadoc/allclasses-noframe.html...
78 Generating ../javadoc/index.html...
79 Generating ../javadoc/help-doc.html...
80 Generating ../javadoc/styleSheet.css...
81 16 warnings
82 moose@pc07:~/Downloads/prog-ws1213$
```

-  ist ein sicheres Zeichen, dass was bei der Zeichenkodierung schief ging.
- Bitte verwendet **immer** UTF-8!
- Eclipse:   


-  ist ein sicheres Zeichen, dass was bei der Zeichenkodierung schief ging.
- Bitte verwendet **immer** UTF-8!

■ Eclipse: Window » Preferences » General » Workspace » Text file encoding  
UTF-8

-  ist ein sicheres Zeichen, dass was bei der Zeichenkodierung schief ging.
- Bitte verwendet **immer** UTF-8!
- Eclipse: 

```
graph LR; Window --> Preferences; Preferences --> General; General --> Workspace; Workspace --> Text_file_encoding[Text file encoding]; UTF_8[UTF-8] --- Text_file_encoding;
```

# Konvention: Leerzeichen

Keine Whitespaces nach

- (Bitweises Komplement)
- ! (Logisches Komplement)
- ++ (Prefix-Inkrementierung, z.B. ++i;)
- - (Prefix-Dekrementierung, z.B. -i;)
- . (Punkt)
- - (Unäres Minus, z.B. -5)
- + (Unäres Plus, z.B. +4)

Und

- Exakt eines vor und nach „=“
- Um Operatoren herum:

```
int i = 42;  
int k = (i * i) / (42 % 3);  
for (int j = 12; j < i; i++) {  
  
}
```

Vorstellung:

- 1D: Vektor, Liste
- 2D: Matrix, Tabelle
- 3D: Quader
- 4D: Hyperwürfel (falls quadratisch)

```
int[] liste = new int[7];  
liste[5] = 5;
```

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

```
int[] [] [] quader = new int[5][7][2];  
quader[0][0][0] = 0;
```

## Was ihr können solltet:

- **Einfache Probleme modellieren:**

Welche Klassen / Methoden brauche ich?

- **Kontrollstrukturen:**

- `if (<Bedingung>) { ... }`

- `while (<Bedingung>) { ... }`

- `for (<Initialisierung>, <Bedingung>, <Update>) { ... }`

- `switch (<Variable>) { case <Wert>: }`

- **Arrays:** 1D- und 2D

- **Kommentare**

- **Koventionen:** Javadoc, Leerzeichen-Setzung

- **Debuggen:** Einfache Fehler in eurem Code finden

## Was ihr hier noch lernt:

- Verwendung der Java Standardbibliothek

- Eingabe von Daten

# Quiz: For-Schleifen (1/2)

```
_____ QuizFor.java _____  
1 public class QuizFor {  
2     public static void main(String[] args) {  
3         int i = 10;  
4         for (; i < 10; i++) {  
5             System.out.println(i);  
6         }  
7         System.out.println("end");  
8     }  
9 }
```

- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?
- Gibt es eine Ausgabe? Welche?



# Quiz: For-Schleifen (1/2) - Antwort

```
_____ QuizFor.java _____  
1 public class QuizFor {  
2     public static void main(String[] args) {  
3         int i = 10;  
4         for (; i < 10; i++) {  
5             System.out.println(i);  
6         }  
7         System.out.println("end");  
8     }  
9 }
```

Ausgabe: **end**, da die Bedingung auch am Anfang überprüft wird

# Quiz: For-Schleifen (2/2)

```
QuizFor.java
1 public class QuizFor {
2     public static void main(String[] args) {
3         int i = 0;
4         for (;;) {
5             System.out.println(i + " bottles of beer");
6             i++;
7         }
8     }
9 }
```

- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?
- Gibt es eine Ausgabe? Welche?

# Quiz: For-Schleifen (2/2) - Antwort

```
_____ QuizFor.java _____  
1 public class QuizFor {  
2     public static void main(String[] args) {  
3         int i = 0;  
4         for (;;) {  
5             System.out.println(i + " bottles of beer");  
6             i++;  
7         }  
8     }  
9 }
```

Ausgabe: Endlosschleife

0 bottles of beer

1 bottles of beer

2 bottles of beer

⋮

## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per `System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$

## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per `System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$

## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per  
`System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$

## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per  
`System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$

## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per  
`System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$



## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per  
`System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$

## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per  
`System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$

## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per  
`System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$

## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per  
`System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$

## Hinweise

- Auf **offizieller Lösung** aufbauen (Verpflichtend!)
- Auf Leerzeichen, gute Variablennamen und Konventionen achten
- Wird eine Bedingung von einem Setter-Parameter nicht eingehalten, schreibt ihr den Wert nicht
  - Stattdessen: Fehlermeldung per  
`System.out.println("dies und das ist falsch");` ausgeben
  - Das ist nur eine Hilfslösung, weil ich noch keine Exceptions hattet
  - Später: (Fast) immer Exceptions!
- Genauigkeit:
  - Positiv bedeutet:  $> 0$
  - Negativ bedeutet:  $< 0$
  - nicht-negativ bedeutet:  $\geq 0$

- 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: Abschlussprüfunsvorbereitung
- 0. 04.02.2013: Abschlussprüfunsvorbereitung
  - 10.02.2013: Ende der Vorlesungszeit des WS 2012/2013 (**Quelle**)

# Vielen Dank für eure Aufmerksamkeit!

