

Java

Scopes, Reference & JavaDoc

Marcus Köhler

15. Januar 2019

Java-Kurs

1. Scopes

2. Die Reference

Intro

Mit der Reference arbeiten

3. JavaDoc

Was ist JavaDoc?

JavaDoc schreiben

Das JavaDoc-Tool

Übung

Scopes

Variablen sind nur innerhalb des Kontexts(d.h. innerhalb der `{ }`) verfügbar, in denen sie deklariert wurden.

Variablen sind nur innerhalb des Kontexts(d.h. innerhalb der `{ }`) verfügbar, in denen sie deklariert wurden.
Das ist alles.

Scopes

Variablen sind nur innerhalb des Kontexts(d.h. innerhalb der `{ }`) verfügbar, in denen sie deklariert wurden.

Das ist alles.

Richtig:

```
1 public class Scope {  
2     int a = 5;  
3     if(a <= 6) {  
4         float f = (float)a;  
5         System.out.println(f  
6         /1.25f);  
7     }  
8     System.out.println(a*10);  
9 }
```

Falsch:

```
1 public class Scope {  
2     int a = 5;  
3     if(a == 6) {  
4         float f = (float)a;  
5     }  
6     System.out.println(f/1.25f);  
7     //f is only available inside  
8     the if block  
9     System.out.println(a*10);  
10 }
```

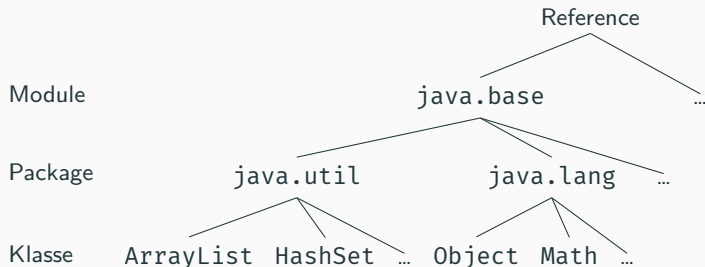
Die Reference

Die Java-Standardbibliothek ist in der sogenannten *Reference* dokumentiert. Jede Java-Version hat ihre eigene Version der Reference; die momentan aktuellste Version (Java 10) ist unter <https://docs.oracle.com/javase/10/docs/api/overview-summary.html> zu finden.

Die Java-Reference

Die Java-Standardbibliothek ist in der sogenannten *Reference* dokumentiert. Jede Java-Version hat ihre eigene Version der Reference; die momentan aktuellste Version (Java 10) ist unter <https://docs.oracle.com/javase/10/docs/api/overview-summary.html> zu finden.

Die Reference ist hierarchisch gegliedert:



Die Seite einer einzelnen Klasse(oder eines Interfaces) in der Reference ist folgendermaßen strukturiert:

1. Ein „Stammbaum“, d.h. wovon geerbt wird, wer erbt, was implementiert wird etc.

Die Seite einer einzelnen Klasse(oder eines Interfaces) in der Reference ist folgendermaßen strukturiert:

1. Ein „Stammbaum“, d.h. wovon geerbt wird, wer erbt, was implementiert wird etc.
2. Eine Beschreibung, wozu die Klasse/das Interface dient

Die Seite einer einzelnen Klasse(oder eines Interfaces) in der Reference ist folgendermaßen strukturiert:

1. Ein „Stammbaum“, d.h. wovon geerbt wird, wer erbt, was implementiert wird etc.
2. Eine Beschreibung, wozu die Klasse/das Interface dient
3. Ein Überblick über Attribute, Konstruktoren und Methoden, die `public` sind

Die Seite einer einzelnen Klasse(oder eines Interfaces) in der Reference ist folgendermaßen strukturiert:

1. Ein „Stammbaum“, d.h. wovon geerbt wird, wer erbt, was implementiert wird etc.
2. Eine Beschreibung, wozu die Klasse/das Interface dient
3. Ein Überblick über Attribute, Konstruktoren und Methoden, die `public` sind
4. Detaillierte Beschreibungen von Konstruktoren und Methoden

Meistens ist das gesuchte in den detaillierten Methodenbeschreibungen zu finden. Darin stehen u.a.:

- Die Parameter, die die Methode braucht

Meistens ist das gesuchte in den detaillierten Methodenbeschreibungen zu finden. Darin stehen u.a.:

- Die Parameter, die die Methode braucht
- Was eine Methode zurück gibt

Meistens ist das gesuchte in den detaillierten Methodenbeschreibungen zu finden. Darin stehen u.a.:

- Die Parameter, die die Methode braucht
- Was eine Methode zurück gibt
- Welche Exceptions eine Methode generieren kann

Die Java-Reference

Meistens ist das gesuchte in den detaillierten Methodenbeschreibungen zu finden. Darin stehen u.a.:

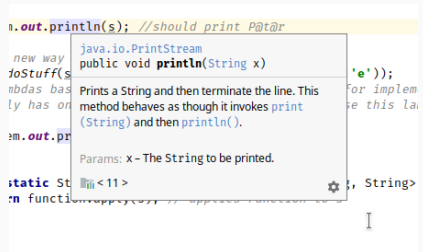
- Die Parameter, die die Methode braucht
- Was eine Methode zurück gibt
- Welche Exceptions eine Methode generieren kann

Viele IDEs haben Möglichkeiten, die detaillierte Beschreibung einer Methode direkt anzuzeigen:

IntelliJ: **Ctrl+Q**

Eclipse: **F2**

Dabei muss der Cursor über dem Element sein, nachdem gesucht werden soll.



Wäre doch auch praktisch, das für
seinen eigenen Code zu haben, oder?

JavaDoc

Was ist JavaDoc?

JavaDoc(**Java Documentation**) ist ein Tool aus dem JDK, welches automatisch bestimmte Kommentare über Methoden, Konstruktoren und Attributen in das gleiche Format wie die Java-Reference umwandelt.

Was ist JavaDoc?

JavaDoc(**Java Documentation**) ist ein Tool aus dem JDK, welches automatisch bestimmte Kommentare über Methoden, Konstruktoren und Attributen in das gleiche Format wie die Java-Reference umwandelt. JavaDoc-Kommentare sind spezielle mehrzeilige Kommentare(`/*` Kommentar `*/`), wobei der Anfang zwei Sterne (`/**`) am Anfang hat:

```
1  /**
2   * Calculates the sum of two {@code ints}.
3   *
4   * @param a The first number of the equation.
5   * @param b The second number of the equation.
6   * @return A {@code long} that represents the sum of the two
7   *         numbers.
8   */
9  public long sumOfInts(int a, int b) { ... }
```

JavaDoc hat ein paar spezielle Annotationen, die sogenannten *Tags*. Diese dienen der Formatierung und beginnen immer mit @.

JavaDoc hat ein paar spezielle Annotationen, die sogenannten *Tags*.
Diese dienen der Formatierung und beginnen immer mit @.
Einige oft verwendete Tags:

@author	Gibt den Autor an
@param <name>	Beschreibung v. Param. <name> (nur Methoden)
@return	Beschreibung des Rückgabewertes (nur Methoden)
@throws <exception>	Wann <exception> auftritt (nur Methoden)
{@code <text>}	Formatiert <text> als Code(d.h. monospaced)

JavaDoc hat ein paar spezielle Annotationen, die sogenannten *Tags*. Diese dienen der Formatierung und beginnen immer mit @. Einige oft verwendete Tags:

@author	Gibt den Autor an
@param <name>	Beschreibung v. Param. <name> (nur Methoden)
@return	Beschreibung des Rückgabewertes (nur Methoden)
@throws <exception>	Wann <exception> auftritt (nur Methoden)
{@code <text>}	Formatiert <text> als Code(d.h. monospaced)

Außer den JavaDoc-Tags kann man den Text auch mit einfachen HTML-Tags(wie <p>, <i> etc.) formatieren.

Um aus einer Quelldatei eine funktionierende Dokumentation zu generieren, muss das Programm `javadoc` aufgerufen werden. Ein Aufruf von `javadoc` kann folgendermaßen aussehen:

```
1 # -d definiert das Zielverzeichnis
2 javadoc org.test.example org.test.database -d ./doc/ #packages
3
4 javadoc JavadocExample.java JavadocExercise.java -d ./doc/ #
   sourcefiles
5
6 javadoc [package] [quelldateien] [optionen] #allgemein
```

Lade dir die Datei JavadocExercise.java aus dem Kurs-Repo herunter und vervollständige das Javadoc. Der genaue Inhalt sollte dabei die Funktion wiedergeben.

- Eine Zusammenfassung und Wiederholung der Kurse seit der letzten Wiederholung

- Eine Zusammenfassung und Wiederholung der Kurse seit der letzten Wiederholung
- Ein größeres praktisches Projekt(2 Termine)

- Eine Zusammenfassung und Wiederholung der Kurse seit der letzten Wiederholung
- Ein größeres praktisches Projekt(2 Termine)
- Ein kurzer Einblick hinter die Kulissen von Java(sehr theoretisch)

- Eine Zusammenfassung und Wiederholung der Kurse seit der letzten Wiederholung
- Ein größeres praktisches Projekt(2 Termine)
- Ein kurzer Einblick hinter die Kulissen von Java(sehr theoretisch)
- Eine Einführung zu Buildsystemen

- Eine Zusammenfassung und Wiederholung der Kurse seit der letzten Wiederholung
- Ein größeres praktisches Projekt(2 Termine)
- Ein kurzer Einblick hinter die Kulissen von Java(sehr theoretisch)
- Eine Einführung zu Buildsystemen
- AuD-Inhalte in Java umgesetzt(z.B. Bäume, Quicksort, Dijkstra)

- Eine Zusammenfassung und Wiederholung der Kurse seit der letzten Wiederholung
- Ein größeres praktisches Projekt(2 Termine)
- Ein kurzer Einblick hinter die Kulissen von Java(sehr theoretisch)
- Eine Einführung zu Buildsystemen
- AuD-Inhalte in Java umgesetzt(z.B. Bäume, Quicksort, Dijkstra)
- Was sind Unit Tests und wie benutze ich sie?

- Eine Zusammenfassung und Wiederholung der Kurse seit der letzten Wiederholung
- Ein größeres praktisches Projekt(2 Termine)
- Ein kurzer Einblick hinter die Kulissen von Java(sehr theoretisch)
- Eine Einführung zu Buildsystemen
- AuD-Inhalte in Java umgesetzt(z.B. Bäume, Quicksort, Dijkstra)
- Was sind Unit Tests und wie benutze ich sie?
- Vorschläge von euch