Java

Scopes, Reference & JavaDoc

Marcus Köhler

11. Januar 2019

Java-Kurs

Überblick

1. Scopes
2. Die Reference
   - Intro
   - Mit der Reference arbeiten
3. JavaDoc
   - Was ist JavaDoc?
   - JavaDoc schreiben
   - Das JavaDoc-Tool
   - Übung

Scopes

Scopes

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

Scopes

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
      /1.25f);
6     }
7     System.out.println(a*10);
8 }
```

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
       the if block
8     System.out.println(a*10);
9 }
```

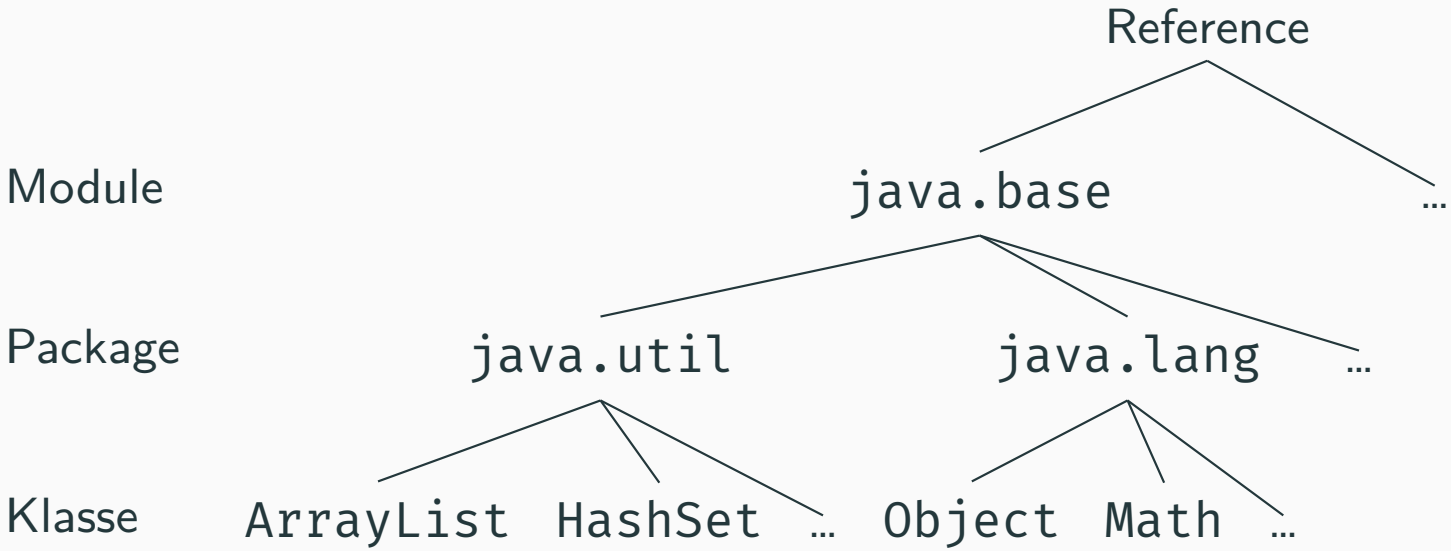
Die Reference

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 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:



Struktur einer Reference-Seite

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.

Struktur einer Reference-Seite

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

Struktur einer Reference-Seite

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

Struktur einer Reference-Seite

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

Die Java-Reference

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

- Die Parameter, die die Methode braucht

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

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

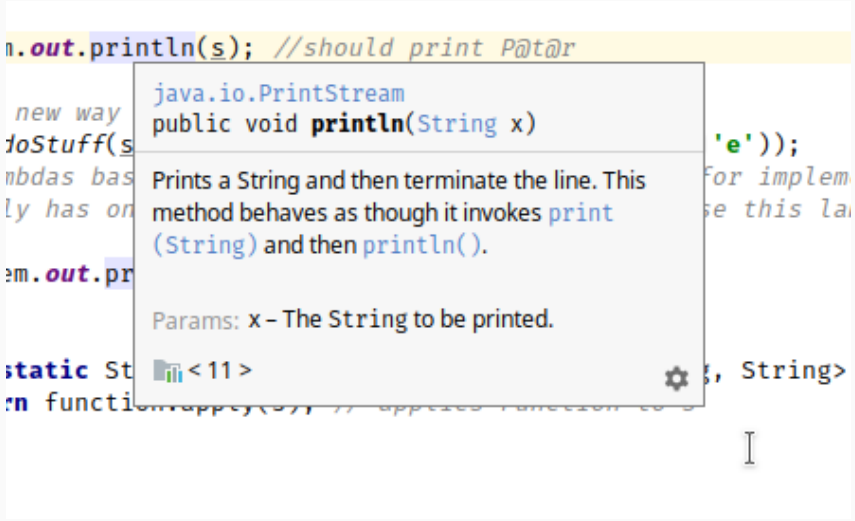
Die Java-Reference

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 Doc**umentation) 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 Doc**umentation) 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
       numbers.
7  */
8 public long sumOfInts(int a, int b) { ... }
```

JavaDoc-Tags

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

JavaDoc-Tags

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-Tags

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.

Das JavaDoc-Tool

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

```
# -d definiert das Zielverzeichnis
javadoc org.test.example org.test.database -d ./doc/ #packages

javadoc JavadocExample.java JavadocExercise.java -d ./doc/ #
    sourcefiles

javadoc [package] [quelldateien] [optionen] #allgemein
```

Übung

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

Weitere Themen

- Eine Zusammenfassung und Wiederholung der Kurse seit der letzten Wiederholung

Weitere Themen

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

Weitere Themen

- 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)

Weitere Themen

- 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 Einfühung zu Buildsystemen

Weitere Themen

- 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 Einfühung zu Buildsystemen
- AuD-Inhalte in Java umgesetzt(z.B. Bäume, Quicksort, Dijkstra)

Weitere Themen

- 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 Einfühung zu Buildsystemen
- AuD-Inhalte in Java umgesetzt(z.B. Bäume, Quicksort, Dijkstra)
- Was sind Unit Tests und wie benutze ich sie?

Weitere Themen

- 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 Einfühung zu Buildsystemen
- AuD-Inhalte in Java umgesetzt(z.B. Bäume, Quicksort, Dijkstra)
- Was sind Unit Tests und wie benutze ich sie?
- Vorschläge von euch