Neues in Java 8

Thomas Darimont

11.07.2013

Hackerspace TKS e.V.

St.-Josef-Straße 8, 66115 Saarbrücken

Historie - Entwicklung von Java

- 1995 **JDK Alpha/Beta**
- 1996 **JDK 1.0** Codename Oak
- 1997 **JDK 1.1** Inner Classes, AWT, JavaBeans, JDBC, RMI, Reflection
- 1998 **J2SE 1.2** Swing, JIT, Collections, CORBA
- 2000 **J2SE 1.3** HotSpot, Java Sound, JNDI
- 2002 **J2SE 1.4** assert, NIO, Logging, <XML/>, WebStart, exception chaining
- 2004 **J2SE 5.0** Generics<?>, @annotations, Autoboxing,Enums, Varargs..., for:each, static imports, j.util.concurrent
- 2006 **Java SE 6** Rhino, Performance improvements, Concurrent GC
- 2011 **Java SE 7** project coin, diamonds, ARM, invoke dynamic multi-catch, Method-Handles binary literals, G1 GC

Java 8

Umfangreichstes Release seit Java 5

Zahlreiche Neuerungen:

- Sprache
- Library
- JVM
- Tooling

Download: https://jdk8.java.net/download.html

Java 8 - Zeitplan

[1] Schedule and status

2012/04/26 **M1**
2012/06/14 **M2**
2012/08/02 **M3**
2012/09/13 **M4**
2012/11/29 **M5**
2013/01/31 **M6**
2013/06/13 **M7** Feature Complete <- Wir sind hier!
2013/09/05 **M8** Developer Preview
2014/01/23 **M9** Final Release Candidate
2014/03/18 **GA** General Availability

JDK 8 is Feature Complete as of build 94, promoted on 2013/6/13.

Java 8 - Neue Funktionaliät

Java 8 Feature Liste (Auszug)

- Lambda Expressions
- Default Methods
- Method References
- Annotation Enhancements
- Bye Permgen! Hello Metaspace!
- Statisch gebundene JNI-Libs
- JSR 310 Date Time API
- Nashorn Javascript Engine

Einen Überblick zu den Neuerungen aus Entwickler Perspektive gibts unter [1]

[0] http://openjdk.java.net/projects/jdk8/features
[1] http://www.techempower.com/blog/2013/03/26/everything-about-java-8/

Lambda Expressions - JEP

JEP 126: Lambda Expressions & Virtual Extension Methods

[1] Summary

Add lambda expressions (closures) and supporting features, including method references, enhanced type inference, and virtual extension methods, to the Java programming language and platform.

[2] Lambdas and closures — what's the difference?

A closure is a lambda expression paired with an environment that binds each of its free variables to a value. In Java, lambda expressions will be implemented by means of closures, so the two terms have come to be used interchangeably in the community.

[3] Gute Übersicht zu Lambda-Expressions "Lambda-FAQ" (Maurice Naftalin)

[1] http://openjdk.java.net/jeps/126
[2] http://www.lambdafaq.org/lambdas-and-closures-whats-the-difference/
[3] http://www.lambdafaq.org/

Lambda Expressions - Erklärung

[0] What is a lambda expression?

In mathematics and computing generally, a lambda expression is a function: for some or all combinations of input values it specifies an output value. Lambda expressions in Java introduce the idea of functions into the language. In conventional Java terms lambdas can be understood as a kind of anonymous method with a more compact syntax that also allows the omission of modifiers, return type, and in some cases parameter types as well.

Syntax

The basic syntax of a lambda is either

```
(parameters) -> expression
```

or

```
(parameters) -> { statements; }
```

Auf github findet man unter [1] ein Lehrreiches Lambda der AdoptOpenJDK Initiative der LJC.

[0] http://www.lambdafaq.org/what-is-a-lambda-expression/
[1] https://github.com/AdoptOpenJDK/lambda-tutorial

Lambda Expressions - Beispiel 0

[0] Examples

```
1. (int x, int y) -> x + y                          // takes two integers and returns their sum
2. (x, y) -> x - y                                  // takes two numbers and returns their
difference
3. () -> 42                                         // takes no values and returns 42
4. (String s) -> System.out.println(s)              // takes a string, prints its value to the
console, and returns nothing
5. x -> 2 * x                                       // takes a number and returns the result of
doubling it
6. c -> { int s = c.size(); c.clear(); return s; }  // takes a collection, clears it, and
returns its previous size
```

[0] http://www.lambdafaq.org/what-is-a-lambda-expression/

Lambda Expressions - Beispiel 1

Aufgabe: "Setze die Farbe *newColor* für alle Formen in der Liste *shapes*".

<= Java 1.4: Hier musste man noch manuell über die Liste iterieren.

```
public static void colorAll(List shapes, Color newColor) {
    for (Iterator itr = shapes.iterator(); itr.hasNext();) {
        Shape s = (Shape)itr.next();
        s.setColor(newColor);
    }
}
```

>= Java 5: Mit der Advanced For-Loop ging das schon einfacher.

```
public static void colorAll(List<Shape> shapes, Color newColor) {
    for (Shape s: shapes) { //Advanced for-Loop Java 5
        s.setColor(newColor);
    }
}
```

>= Java 8: Mit Lambda-Expression: Das WAS statt WIE steht im Vordergrund!

```
public static void colorAll(List<Shape> shapes, Color newColor) {
    shapes.forEach(s -> s.setColor(newColor));   //Lambda Expression Java 8
}
```

Lambda Expressions - Beispiel 2

DEMO

Default Methods

Erlaubt die Implementierung von default-Verhalten an Interface-Methoden.

Beispiel: java.lang.Iterable#forEach

```
public default void forEach(Consumer<? super T> action) {
    Objects.requireNonNull(action);
    for (T t : this) {
        action.accept(t);
    }
}
```

Erlaubt die nachträgliche Erweiterung von Klassenbibliotheken ohne die Verwender des alten API zur Implementierung von neuen Methoden zu zwingen.
Alter Code kompiliert auch noch mit neuem API.
Bestehende Methoden haben Vorrang vor default-Methoden.

DEMO

Method References

[0] Übersicht der Varianten:

Static method reference: String::valueOf

Non-static method reference: Object::toString

Capturing method reference: x::toString

Constructor reference: ArrayList::new

[1] Beispiele für Varianten:

Method reference	Equivalent lambda expression
String::valueOf	x -> String.valueOf(x)
Object::toString	x -> x.toString()
x::toString	() -> x.toString()
ArrayList::new	() -> new ArrayList<>()

[0,1] http://www.techempower.com/blog/2013/03/26/everything-about-java-8/

Annotation Enhancements

[0] Type Annotations

Basis für flexible Type Checkers. Siehe auch Prof. Michael Ernst Checker Framework [1]

[2] Repeating Annotations:

Annotation können wiederholt verwendet werden.

Neue Meta-Annotation: @Repeatable

```
@A("Cool")
@A("Hip")
@Component
class CoolAndHipComponent {
   ...
}
```


[0] http://openjdk.java.net/projects/type-annotations/

[1] http://homes.cs.washington.edu/~mernst/software/#checker-framework

[2] http://docs.oracle.com/javase/tutorial/java/annotations/repeating.html

Bye Permgen! Hello Metaspace!

Permgen

[0] Im Permgen werden JVM interne Strukturen / Reflektion Artefakte und interned String Literale abgelegt. [1]Seit Java SE 7 liegen interned Strings im normalen Java Heap.

Metaspace

[2] In Java SE 8 wird der Permgen abgeschafft. Dafür wird der Metaspace eingeführt. Der Metaspace ist nativer Off-Heap Speicher, in dem Metadaten zu Klassen abgelegt werden. Parameter zum Memory-Sizing: MaxMetaspaceSize (optional). Metaspace wird gc'ed (wenn MaxMetaspaceSize erreicht). PermSize Speicher konfigurationsoptionen werden ignoriert!

[0] https://blogs.oracle.com/jonthecollector/entry/presenting_the_permanent_generation

[1] http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html#jdk7changes

[2] http://javaeesupportpatterns.blogspot.de/2013/02/java-8-from-permgen-to-metaspace.html

Statisch gebundene JNI Libs

[0] ... eine der am stärksten Unterschätzten Änderungen.

Derzeit verhindern dynamisch gebundene JNI Libs die Verwendung einer JVM auf Apples IOS Betriebssystem. Durch statisch gebundene Bibliotheken in Java kann eine JVM auch unter den IOS Beschränkungen laufen [1].

[0] http://openjdk.java.net/jeps/178

[1] http://www.infoworld.com/t/java-programming/gosling-new-java-proposal-could-ease-ports-ios-213843

JSR 310 / JEP 150 Date Time API

Einfaches API für Datums - / Zeitarithmetik [0] javax.time.*

Soll Strukturen um java.util.Date, Gregorian Calendar etc. ablösen. Setzt auf ein fluent API, Value Objects, Immutabillity, etc.

Beispiele findet man in den Tests zu JSR 310 [1]

[0] http://openjdk.java.net/jeps/150

[1] https://github.com/ThreeTen/threeten/tree/0b071a60997f409e44b9bbccde013b004f24fe22/src/test/java/javax/time

Nashorn Javascript Engine

[0] Spezielle Implementierung einer Java Script Engine, welche invoke dynamic verwendet.

[1] Generiert bytecode on-the-fly, großer Performance Boost durch Verwendung von invoke dynamic in der JVM.

[0] http://openjdk.java.net/projects/nashorn/

[1] http://openjdk.java.net/jeps/174

Fragen?

... es gibt natürlich noch sehr viel mehr neues in Java 8.

Ich freue mich die Neuerungen gemeinsam mit euch zu entdecken :)