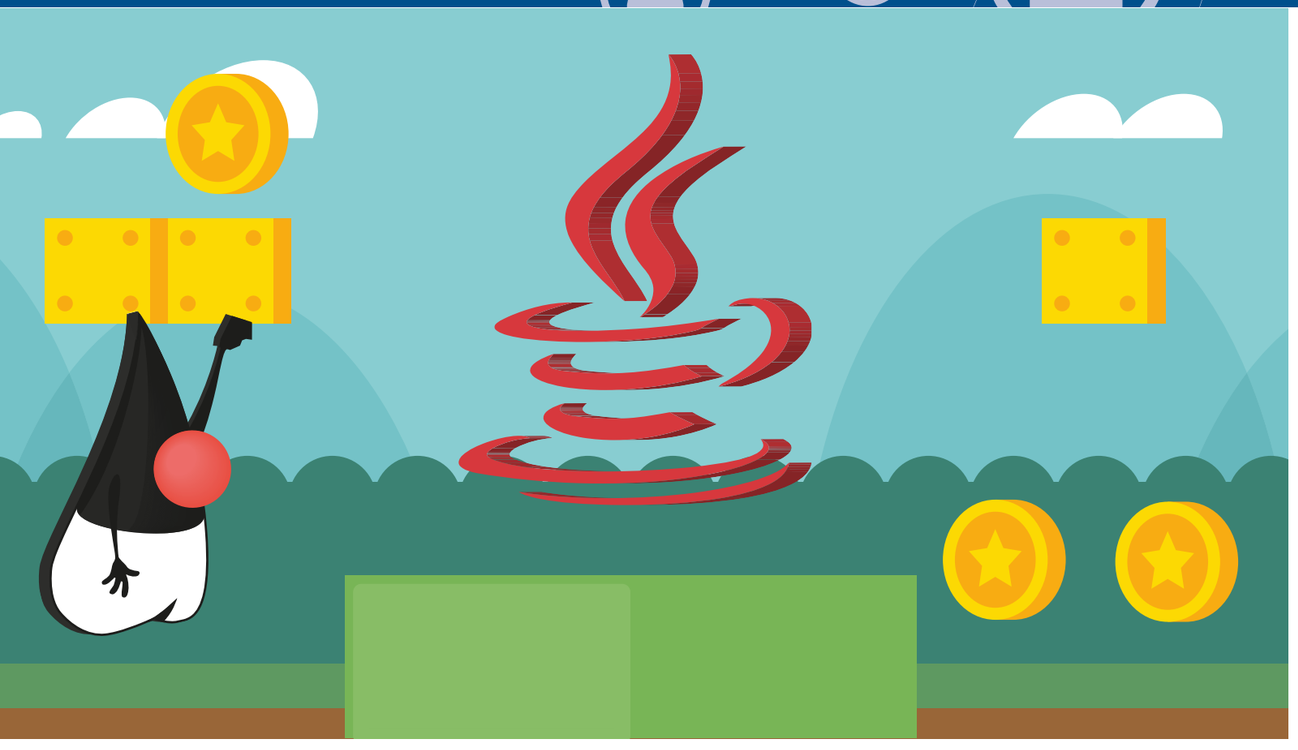

Version 11: Next Level Java

Falk Sippach, Orientation in Objects GmbH

Pünktlich sechs Monate nach Java 10 ist Ende September bereits Java 11 [1] erschienen. Allerdings waren die letzten Wochen und Monate stark geprägt von den Debatten zu Oracles neuer Support- und Lizenzpolitik sowie der Frage, ob Java beziehungsweise das JDK überhaupt kostenlos bleiben. Die Informationen zu den neuen Features und die Änderungen an der Syntax sowie der Klassen-Bibliothek gingen dabei etwas unter. Dieser Artikel beleuchtet beide Themen näher.

Mit Java 11 führt Oracle eine neue Lizenz- und Support-Strategie ein. Freie Updates für die letzte Long-Term-Support-Version (LTS-Version) 8 wird es ab Januar 2019 nicht mehr geben. Zudem ist ab Java 11 das Oracle JDK nur noch in der Entwicklung kostenlos nutzbar, in Produktion muss ein Support-Vertrag *[2]* mit Oracle gegen entsprechende Gebühren abgeschlossen werden.

Allerdings ist das Oracle JDK ab sofort binär kompatibel mit dem OpenJDK, das als Open Source unter der GNU General Public License v2 (with the Classpath Exception - GPLv2+CPE) veröffentlicht ist. Um das JDK weiterhin in Produktion kostenlos einsetzen zu können, kann man also ab Java 11 das OpenJDK verwenden. Allerdings bietet Oracle immer nur Updates für die aktuelle OpenJDK-Version; mit dem nächsten halbjährlichen Major-Release muss man also potenziell die Anwendung bereits auf die nächste Version updaten.

Alternative Anbieter wie Azul und IBM haben im Rahmen des AdoptOpenJDK-Projekts allerdings bereits angekündigt, die LTS-Versionen (etwa Java 11) auch bis zu vier Jahre mit kostenlosen Updates zu versorgen. Auch für Kunden von diversen Linux-Distributionen (wie Red Hat Enterprise Linux) wird das OpenJDK 8 noch mindestens vier Jahre im Rahmen der Betriebssystem-Support-Verträge kostenlos mit Updates versorgt. Für Anwender der Java-Plattform gibt es also genügend Optionen zwischen kommerziellen Lösungen von Oracle (und auch Azul Zulu, IBM SDK etc.) sowie weiterhin freien Varianten mit dem OpenJDK und dem verlängerten Support des AdoptOpenJDK-Projekts *[3]* oder im Rahmen anderer Lizenzvereinbarungen (Red Hat Enterprise Linux *[4]*, IBM SDK *[5]* etc.). Allerdings gehört etwas Mut dazu, Oracle den Rücken zu kehren und alternativen Java-Plattformen eine Chance zu geben.

Neue Features

Die Neuerungen von Java 11 fallen relativ übersichtlich aus. Das verwundert wegen des kurzen Zeitraums seit der Veröffentlichung der vorangegangenen Version nicht weiter. Trotzdem wurden insgesamt mehr als 2.400 Tickets geschlossen *[6]*. Fast 80 Prozent davon hat natürlich Oracle bearbeitet, an den restlichen 20 Prozent waren viele andere Firmen wie SAP, Red Hat, Google oder IBM beteiligt. Ein Großteil der Neuerungen wurde im Rahmen der folgenden Java Enhancement Proposals (JEPs) umgesetzt:

- 181: Nest-Based Access Control
- 309: Dynamic Class-File Constants
- 315: Improve Aarch64 Intrinsics
- 318: Epsilon: A No-Op Garbage Collector
- 320: Remove the Java EE and CORBA Modules
- 321: HTTP Client (Standard)
- 323: Local-Variable Syntax for Lambda Parameters
- 324: Key Agreement with Curve25519 and Curve448
- 327: Unicode 10
- 328: Flight Recorder
- 329: ChaCha20 and Poly1305 Cryptographic Algorithms
- 330: Launch Single-File Source-Code Programs
- 331: Low-Overhead Heap Profiling
- 332: Transport Layer Security (TLS) 1.3
- 333: ZGC: A Scalable Low-Latency Garbage Collector
- 335: Deprecate the Nashorn JavaScript Engine
- 336: Deprecate the Pack200 Tools and API

Aus Entwicklersicht sind nur einige wenige Punkte wirklich relevant. So wurde im JEP 323 eine Erweiterung der in Java 10 eingeführten Local-Variable Type Inference umgesetzt. Type Inference ist das Schlussfolgern der Datentypen aus den restlichen Angaben des Quellcodes und den Typisierungsregeln heraus. Das spart Schreibarbeit und bläht den Quellcode nicht unnötig auf, wodurch sich wiederum die Lesbarkeit erhöht. Seit Java 10 können lokale Variablen mit dem Schlüsselwort „var" folgendermaßen deklariert werden *(siehe Listing 1)*.

```
// Funktioniert seit Java 10
var zahl = 5; // int
var string = "Hello World"; // String
var objekt = BigDecimal.ONE; // BigDecimal
```

Listing 1

Neu in Java 11 ist, dass man nun auch Lambda-Parameter mit „var" deklarieren kann. Das mag auf den ersten Blick nicht sonderlich sinnvoll erscheinen, da man den Typ von Lambda-Parametern sowieso weglassen und über die Typ-Inferenz ermitteln lassen kann. Nützlich wird die Erweiterung aber für die Verwendung von Type Annotations wie „@Nonnull" und „@Nullable" *(siehe Listing 2)*.

```
// Inference von Lambda Parametern
Consumer<String> printer = (var s) -> System.out.println(s); // statt s -> System.out.println(s);

// aber keine Mischung von "var" und deklarierten Typen möglich
// BiConsumer<String, String> printer = (var s1, String s2) -> System.out.println(s1 + " " + s2);

// Nützlich für Type Annotations
BiConsumer<String, String> printer = (@Nonnull var s1, @Nullable var s2) -> System.out.println(s1 + (s2 == null ? ""
: " " + s2));
```

Listing 2

Die nächste interessante Neuerung ist die Standardisierung des bisher noch experimentellen neuen HTTP-Client-API, das mit JDK 9 eingeführt und in JDK 10 aktualisiert wurde (JEP 110). Neben HTTP/1.1 werden nun auch HTTP/2, WebSockets, HTTP/2 Server Push, synchrone und asynchrone Aufrufe sowie Reactive Streams unterstützt. Garniert mit einem gut lesbaren Fluent-Interface wird die Verwendung von anderen HTTP-Clients (wie Apache) dann in Zukunft voraussichtlich obsolet sein *(siehe Listing 3)*.

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
  .uri(URI.create("http://openjdk.java.net/"))
  .build();
client.sendAsync(request, asString())
  .thenApply(HttpResponse::body)
  .thenAccept(System.out::println)
  .join();
```

Listing 3

Durch den JEP 330 (Launch Single-File Source-Code Programs) lassen sich jetzt Klassen starten, die noch nicht kompiliert wurden. Programme mit einer einzigen Datei sind heutzutage beim Schreiben kleiner Hilfsprogramme üblich und insbesondere die Domäne von Skript-Sprachen. Nun kann man sich auch in Java die unnötige Arbeit sparen und das verringert zugleich die Einstiegshürde für Java-Neulinge *(siehe Listing 4)*. Auf unixoiden Betriebssystemen können Java-Dateien als Shebang-Files sogar direkt ausgeführt werden *(siehe Listing 5)*.

```
# java HelloWorld.java
// statt
# javac HelloWorld.java
# java -cp . hello.World
```

Listing 4

```
#!/path/to/java --source version
# ./HelloWorld.java
```

Listing 5

Weitere erwähnenswerte Änderungen sind die Unterstützung des Unicode-10-Standards und die Integration der bisher nur mit einer kommerziellen Lizenz verwendbaren Profiling-Tools „Mission Control" und „Flight Recorder" in das OpenJDK (sie wurden bisher nur mit dem Oracle JDK ausgeliefert). Ziel des Flight Recorder ist das möglichst effiziente Aufzeichnen von Anwendungsdaten, um bei Problemen die Java-Anwendung und die JVM analysieren zu können.

Etwas verrückt scheint zunächst der JEP 318 (Epsilon: A No-Op Garbage Collector) zu sein. Dabei handelt es sich um einen neuen Garbage Collector, der aber gar keine Garbage Collection durchführt (deshalb No-Op, also No-Operation). Interessant ist dieses Verhalten aber für Serverless Functions im Rahmen des Oracle-fn-Projekts. Da es sich hier um sehr kurz laufende Java-Anwendungen handelt, wäre ein zwischenzeitlicher Garbage-Collector-Lauf eher kontraproduktiv und sinnlos, wenn die Anwendung kurz darauf sowieso wieder beendet wird.

API-Änderungen

An der Java-Klassenbibliothek gab es natürlich auch unzählige kleine Änderungen. Besonders viel hat sich bei String und Character getan *(siehe Listing 6)*.

```
| Welcome to JShell -- Version 11
| For an introduction type: /help intro
// Unicode zu String
jshell> Character.toString(100)
$1 ==> "d"
jshell> Character.toString(66)
$2 ==> "B"

// Zeichen mit Faktor multiplizieren
jshell> "-".repeat(20)
$3 ==> "--------------------"

// Enthält ein Text keine Zeichen (höchstens Leerzeichen)?
jshell> String msg = "hello"
msg ==> "hello"
jshell> msg.isBlank()
$5 ==> false
jshell> String msg = "   "
msg ==> "   "
jshell> msg.isBlank()
$7 ==> true

// Abschneiden von führenden oder nachgelagerten Leerzeichen
jshell> " hello world    ".strip()
$8 ==> "hello world"
jshell> "hello world     ".strip()
$9 ==> "hello world"
jshell> "hello world     ".stripTrailing()
$10 ==> "hello world"
jshell> "hello world     ".stripLeading()
$11 ==> "hello world     "
jshell> "      ".strip()
$12 ==> ""

// Texte zeilenweise verarbeiten
jshell> String content = "this is a multiline content\nMostly obtained from some file\rwhich we will break into
lines\r\nusing the new api"
content ==> "this is a multiline content\nMostly obtained fro ... ines\r\nusing the new api"
jshell> content.lines().forEach(System.out::println)
this is a multiline content
Mostly obtained from some file
which we will break into lines
using the new api
```

Listing 6

Was entfernt wurde

Die Ankündigungen in Form von Deprecations in den Versionen 9 und 10 sind nun in Java 11 Wirklichkeit geworden. Im JEP 320 wurden diverse Java-Enterprise-Packages aus Java SE entfernt, darunter JAX-WS (XML-basierte SOAP-Webservices inklusive der Tools „wsgen" und „wsimport"), JAXB (Java XML Binding inklusive der Tools „schemagen" und „xjc"), JAF (Java-Beans-Activation-Framework), Common Annotations („@PostConstruct", „@Resource" etc.), CORBA und JTA (Java-Transaction-API).

Neu ist auch, dass das Oracle JDK kein JavaFX mehr enthalten wird (mit dem OpenJDK wurde es übrigens noch nie ausgeliefert). Stattdessen wird JavaFX über OpenJFX *[7]* als separater Download angeboten und kann wie jede andere Bibliothek in beliebigen Java-Anwendungen verwendet werden.

Neben JavaFX ist auch der Support für Applets und Java Web Start eingestellt. Die Open-Source-Community plant allerdings bereits Nachfolge-Projekte *[8]*. Wenn man im Moment noch Java Web Start nutzen möchte, muss man zunächst beim Oracle JDK 8 bleiben und entweder ohne Sicherheits-Updates leben oder für den kommerziellen Support ab 2019 Geld ausgeben.

Neu in Java 11 als „Deprecated" markiert wurde die JavaScript-Engine Nashorn. Es ist davon auszugehen, dass sie in zukünftigen Java-Versionen verschwinden wird. Nashorn hat sich allerdings nie so richtig als serverseitige JavaScript-Implementierung gegenüber Node.js durchsetzen können. Mit der GraalVM geht Oracle mittlerweile alternative Wege, um andere Programmiersprachen nativ auf der JVM auszuführen.

Übrigens wird es ab Java 11 die Java-Laufzeitumgebung (JRE) nur noch in der Server-Variante und nicht mehr für Desktops geben. Allerdings kann man für Desktop-Anwendungen mit dem Modulsystem und dem Werkzeug „jlink" mittlerweile selbst sehr einfach in der Größe angepasste Laufzeit-Umgebungen erstellen.

Fazit und Ausblick

Die großen Überraschungen sind ausgeblieben; vielmehr finalisiert Java 11 angefangene Arbeiten aus den beiden vorangegangenen Versionen, damit es als LTS-Release für die nächsten drei Jahre gut gewappnet ist. Dazu wurden auch einige alte Zöpfe abgeschnitten und zum Beispiel JavaFX und diverse Java-EE-Packages aus dem JDK entfernt.

Java 12 steht bereits in den Startlöchern und wird voraussichtlich im März 2019 erscheinen. Die Liste der Neuerungen wächst noch, interessant für Entwickler sind aber insbesondere die folgenden beiden; weitere werden sicher in den nächsten Wochen folgen:

- JEP 325: Switch Expressions
- JEP 326: Raw String Literals

Beide Sprach-Features stammen aus sogenannten „Inkubator-Projekten" (Amber *[9]*, Valhalla *[10]*, Loom *[11]*), in denen kleinere, die Entwicklerproduktivität steigernde Sprach- und VM-Features ausgebrütet und dann als Java Enhancement Proposals (JEPs) akzeptiert werden. Switch Statements können demnach in Zukunft auch als Expression verwendet werden, die das Ergebnis des entsprechenden Case-Zweigs direkt zurückliefert *(siehe Listing 7)*.

Mit Raw String Literals wird Java endlich die Möglichkeit bekommen, mehrzeilige Zeichenketten zu definieren, die zusätzlich Escape-Sequenzen (..) ignorieren. Damit lässt sich viel einfacher mit regulären Ausdrücken und Windows-Dateipfaden umgehen. Einzig das Ersetzen von Variablen (String-Interpolation) ist im Moment noch nicht geplant, ein Feature, das alternative Sprachen wie Groovy, Ruby und JavaScript schon länger unterstützen *(siehe Listing 8)*.

Java-Entwickler können sich also auch in den nächsten Jahren auf viele interessante neue Features freuen. Jetzt gilt es allerdings erstmal, die neuen Funktionen von Java 11 auszuprobieren und sich mit den veränderten Bedingungen bei den Lizenzen und beim Support auseinanderzusetzen.

```
int numLetters = switch (day) {
     case MONDAY, FRIDAY, SUNDAY  -> 6;
     case TUESDAY                 -> 7;
     case THURSDAY, SATURDAY      -> 8;
     case WEDNESDAY               -> 9;
};
```

Listing 7

```
Runtime.getRuntime().exec(`"C:\Program Files\foo" bar`);
// statt
Runtime.getRuntime().exec("\"C:\\Program Files\\foo\" bar");

String script1 = `function hello() {
                  print('"Hello World"');
                  }

                  hello();`;
// statt
String script2 =  "function hello() {\n" +
                  "   print(\'\"Hello World\"\');\n" +
                  "}\n" +
                  "\n" +
                  "hello();\n";
```

Listing 8

Referenzen

[1] JDK-11-Projektseite: *http://jdk.java.net/11/*
[2] Java-SE-Support-Verträge: *https://blogs.oracle.com/java-platform-group/a-quick-summary-on-the-new-java-se-subscription*
[3] AdoptOpenJDK: *https://adoptopenjdk.net*
[4] RHEL OpenJDK: *https://access.redhat.com/articles/1299013*
[5] IBM SDK: *https://developer.ibm.com/javasdk/2018/04/26/java-standard-edition-ibm-support-statement*
[6] OpenJDK-Community-Zusammenarbeit: *https://blogs.oracle.com/java-platform-group/building-jdk-11-together*
[7] OpenJFX: *http://openjdk.java.net/projects/openjfx*
[8] Opensource Java Webstart: *https://dev.karakun.com/webstart*
[9] Project Amber: *http://openjdk.java.net/projects/amber*
[10] Project Valhalla: *http://openjdk.java.net/projects/valhalla*
[11] Project Loom: *http://openjdk.java.net/projects/loom*

Falk Sippach
falk.sippach@oio.de

Falk Sippach hat mehr als fünfzehn Jahre Erfahrung mit Java und ist bei der Mannheimer Firma OIO Orientation in Objects GmbH als Trainer, Software-Entwickler und Projektleiter tätig. Er publiziert regelmäßig in Blogs, Fachartikeln und auf Konferenzen. In seiner Wahlheimat Darmstadt organisiert er mit Anderen die örtliche Java User Group. Falk Sippach twittert unter @sippsack.