1 Einleitung

Während früher die Java-Releases aufgrund unfertiger Features häufig verschoben wurden, hat Oracle mit Java 10 auf halbjährliche Releases umgestellt, die jeweils die bis zu diesem Zeitpunkt fertig implementierten Features bereitstellen. Dadurch wurden sowohl Java 10 als auch Java 11 beide pünktlich im Abstand von rund 6 Monaten veröffentlicht und dies wird vermutlich auch für Java 12 gelten, dessen Releasetermin für den 19. März 2019 geplant ist.

Während diese schnelle Releasefolge eine größere Herausforderung für Toolhersteller ist, kann dies für uns als Entwickler aber positiv sein, weil wir potenziell weniger lang auf neue Features warten müssen. Das konnte früher recht mühsam sein, wie die letzten Jahre gezeigt haben.

Rund 3,5 Jahre nach dem Erscheinen von JDK 8 am 18. März 2014 ging Java mit Version 9 im September 2017 an den Start. Wieder einmal musste die Java-Gemeinde auf die Veröffentlichung der Version 9 des JDKs länger warten – es gab gleich mehrere Verschiebungen, zunächst von September 2016 auf März 2017, dann auf Juli 2017 und schließlich auf September 2017. Aber immerhin hat sich das Warten gelohnt: Neben diversen Verbesserungen im JDK selbst lag bei Java 9 der Hauptfokus auf der Modularisierung, die eine verlässliche Konfiguration und besser strukturierte Programme mit klaren Abhängigkeitsbeziehungen begünstigt.[1]

Was erwartet Sie im Folgenden?

Dieses Buch gibt einen Überblick über diverse wesentliche Erweiterungen in den JDKs 9, 10, 11 und 12. Es werden unter anderem folgende Themen behandelt:

API- und Syntaxerweiterungen Wir schauen uns verschiedene Änderungen an der Syntax von Java an. Neben Erweiterungen bei der `@Deprecated`-Annotation widmen wir uns Details zu Bezeichnern, dem Diamond Operator und vor allem gehe ich kritisch auf das Feature privater Methoden in Interfaces ein. Für Java 10 und 11 thematisiere ich die Syntaxerweiterung `var` als Möglichkeit zur Definition lokaler Variablen bzw. zur Verwendung in Lambdas.

Kommen wir zu den APIs: In Java 9 wurden diverse APIs ergänzt oder neu eingeführt. Auch Bestehendes, wie z. B. das Stream-API oder die Klasse `Optional<T>`,

[1] Allerdings sollte man bedenken, dass sowohl die funktionale Programmierung mit Lambdas als auch die Modularisierung bereits für JDK 7 angekündigt waren.

wurde um Funktionalität ergänzt. Neben Vereinfachungen beim Prozess-Handling, der Verarbeitung mit `Optional<T>` oder von Daten mit `InputStream`s schauen wir auf fundamentale Neuerungen im Bereich der Concurrency durch Reactive Streams. Darüber hinaus enthalten Java 10 und 11 eine Vielzahl kleinerer weiterer Neuerungen. Eine größere Änderung ist der mit Java 11 offiziell ins JDK 11 aufgenommene HTTP/2-Support. Eher schmerzlich könnte der Wegfall verschiedener Funktionalitäten, etwa rund um JAXB sowie JavaFX, aufgenommen werden. In Java 12 wurden leider die sogenannten »Raw String Literals« als Feature gestrichen, sodass für uns als Entwickler vor allem ein Preview auf Syntaxänderungen bezüglich `switch` verbleibt.

JVM-Änderungen In jeweils eigenen Abschnitten beschäftigen wir uns mit Änderungen in der JVM, die in den neuen Java-Versionen enthalten sind, für JDK 9 etwa in Bezug auf die Nummerierung von Java-Versionen oder `javadoc`. Zudem kann für Quereinsteiger und Neulinge die durch das Tool `jshell` bereitgestellte Java-Konsole mit REPL-Unterstützung (Read-Eval-Print-Loop) erste Experimente und Gehversuche erleichtern, ohne dafür den Compiler oder eine IDE bemühen zu müssen. Mit Java 11 kommt ein neuer Garbage Collector, mit dem Flight Recoder ein neues Tool sowie mit dem Feature »Launch Single-File Source-Code Programs« die Möglichkeit, Java-Klassen ohne explizite vorherige Kompilierung ausführen zu lassen und somit für Scripting einsetzen zu können. Mit Java 12 kann man die Integration des Microbenchmark-Frameworks JMH (Java Microbenchmarking Harness) als wesentliche Neuerung ansehen.

Modularisierung Die Modularisierung adressiert zwei typische Probleme größerer Java-Applikationen. Zum einen ist dies die sogenannte JAR-Hell, womit gemeint ist, dass sich im `CLASSPATH` verschiedene JARs mit zum Teil inhaltlichen Überschneidungen (unterschiedliche Versionen mit Abweichungen in Packages oder gleiche Klassen, aber anderem Bytecode) befinden. Dabei kann aber nicht sichergestellt werden, wann welche Klasse aus welchem JAR eingebunden wird. Zum anderen sind als `public` definierte Typen beliebig von anderen Packages aus zugreifbar. Das erschwert die Kapselung. Mit JDK 9 kann man nun eigenständige Softwarekomponenten (Module) mit einer Sichtbarkeitssteuerung definieren. Das hat allerdings weitreichende Konsequenzen: Sofern man Module verwendet, lassen sich Programme mit JDK 9 nicht mehr ohne Weiteres wie gewohnt starten, wenn diese externe Abhängigkeiten besitzen. Das liegt vor allem daran, dass Abhängigkeiten nun beim Programmstart geprüft und dazu explizit beschrieben werden müssen.

Es gibt aber einen rein auf dem `CLASSPATH` basierenden Kompatibilitätsmodus, der ein Arbeiten wie bis einschließlich JDK 8 gewohnt ermöglicht.

Tipp: Beispiele und der Kompatibilitätsmodus

Zum Ausprobieren einiger Neuerungen aus JDK 9, 10, 11 und 12 werden wir kleine Beispielapplikationen in `main()`-Methoden erstellen. Dabei ist es für erste Experimente und für die Migration bestehender Anwendungen von großem Vorteil, dass man das an sich modularisierte JDK auch ohne eigene Module und ihre Sichtbarkeitsbeschränkungen betreiben kann. In diesem Kompatibilitätsmodus wird wie zuvor bei Java 8 mit `.class`-Dateien, JARs und dem `CLASSPATH` gearbeitet. Für zukünftige Projekte wird man aber mitunter Module nutzen wollen. Das schauen wir uns in eigenen Kapiteln an.

Entdeckungsreise JDK 9, 10, 11 und 12 – Wünsche an die Leser

Ich wünsche allen Lesern viel Freude mit diesem Buch sowie einige neue Erkenntnisse und viel Spaß beim Experimentieren mit JDK 9, 10, 11 und 12. Möge Ihnen der Umstieg auf die neue Java-Version und die Erstellung modularer Applikationen oder die Migration bestehender Anwendungen durch die Lektüre meines Buchs leichter fallen.

Wenn Sie zunächst eine Auffrischung Ihres Wissens zu Java 8 und seinen Neuerungen benötigen, bietet sich ein Blick in den Anhang A an.