JAVA 17

Oracle heeft sinds Java SE 10 een 6 maanden release-schema geïnstalleerd, in tegenstelling tot de (zo ongeveer) 3 jaars releases daarvoor. De deadline voor nieuwe features in Java is fixed gemaakt. Dit betekent dat het aantal JDK Enhancement Proposals (JEP) dat geïmplementeerd kan worden per release variabel is geworden. Eens in de drie jaar zal een zogenaamde Long-Term-Support (LTS) versie worden gemaakt. Java SE 17 is zo'n LTS versie en zal door Oracle nog tot september 2026 ondersteund worden. Dit is best een big deal, omdat veel grote bedrijven het beleid hebben één versie achter de laatste LTS te blijven zitten. Een veelgehoord argument is dat de kinderziektes dan opgelost zouden zijn.

In Java SE 17 LTS staan 14 features (JEP's) gepland. Om met wat Java SE 17 features te kunnen spelen, zonder de early access echt te hoeven installeren, is alle code in dit artikel uitgevoerd binnen een Docker container met OpenJDK 17 (listing 1).

L1

```
$ docker run -it --rm \
      -v $(pwd)/src/main/java:/src \
      openjdk:17 /bin/bash
$ cd /src
```

Dit artikel zal in drie hoofdsecties verdeeld worden. Het eerste deel gaat over nieuwe standaard features. Het tweede deel gaat in op preview en incubator features en het derde deel gaat in op het deprecated verklaren of verwijderen van features. Bij elke feature zal het JEP-nummer vermeld worden.

{ NIEUWE STANDAARD FEATURES }

356: Enhanced Pseudo-Random Number Generators

Deze JEP maakt het gemakkelijk om aangepaste pseudorandom number generators (PRNGs) algoritmen te gebruiken en biedt een paar concrete PRNG-implementaties. Het is momenteel moeilijk om een aangepaste PRNG-implementatie te verkrijgen die gemakkelijk java.util.Random kan vervangen, omdat het geen interface is. Deze JEP introduceert een nieuwe interface genaamd `RandomGenerator`. Deze interface levert een uniforme API voor alle bestaande en nieuwe PRNGs. Er worden ook vier gespecialiseerde RandomGenerator interfaces aangeleverd: `SplittableRandomGenerator`, `JumpableRandomGenerator`, `LeapableRandomGenerator`, `ArbitrarilyJumpableRandomGenerator`.

De bestaande Random, ThreadLocalRandom en SplittableRandom zijn gerefactored om netjes binnen deze nieuwe manier te passen en meer toekomstvast te zijn.

Ivo Woltring is Principal Expert en Codesmith bij Ordina JTech en houdt zich graag bezig met nieuwe ontwikkelingen in de software wereld.

382: New macOS Rendering Pipeline

Deze JEP verandert de interne Java 2D-renderingpijplijn van OpenGL naar Apple Metal API. Dit wordt een implementatie wijziging zonder wijziging in API's.

391: macOS/AArch64 Port

Port de JDK naar macOS. Apple heeft in haar lange termijn plannen aangegeven met hun reeks Macintosh computers over te gaan van x64 naar AArch64. Er wordt dus een brede vraag verwacht. Voor Windows is dit in Java 16 al gedaan in JEP 388.

403: Strongly Encapsulate JDK Internals

Langzaam maar zeker worden de JDK Internals verborgen voor misbruik. Het is nooit de bedoeling geweest dat Interne API's gebruikt zouden worden door Java-applicaties, aangezien dit de evolutie en het onderhoud van de interne werking moeilijker maakt. Deze JEP is de opvolger van JEP 396 in Java 16. Het wordt steeds moeilijker gemaakt om de interne API's te blijven gebruiken en het stimuleert ontwikkelaars om te migreren naar alternatieven.

409: Sealed Classes

Deze JEP biedt Java een manier om implementaties en overerving van classes, interfaces en records te beperken tot wat expliciet toegestaan is (Listing 2). In Java SE 15 werd de eerste Preview aangeboden (JEP 360). In Java SE 16 was de tweede Preview (JEP 397) en in Java SE 17 wordt dit een feature zonder verdere wijzigingen ten opzichte van Java SE 16.

L2

```
public sealed class Fruit permits Orange {}

public non-sealed class Orange extends Fruit {}

public class Apple extends Fruit {}

$ javac Fruit.java Orange.java Apple.java
Apple.java:1: error: class is not allowed to
extend sealed class: Fruit (as it is not listed
in its permits clause)
public class Apple extends Fruit {
       ^
1 error
```

415: Context-Specific Deserialization Filters

Deserialiseren van onbekende data is een potentieel gevaarlijke activiteit. De binnenkomende stream bepaalt namelijk welke objecten gemaakt worden, welke waarden ze hebben en bepaalt ook de relaties tussen de gemaakte objecten. Het kan zijn dat het maken van dit soort objecten neveneffecten heeft en kan dus de integriteit ondermijnen.
In Java SE 9 zijn deserialisatie filters ingevoerd (JEP 290) om het mogelijk te maken voor applicaties om de binnenkomende stream eerste te valideren voordat er gedesialiseerd wordt. Dit had beperkingen en deze JEP zorgt ervoor dat deze geadresseerd worden. In plaats van de JVM brede deserialisatie filter aan te roepen wordt er vanaf nu de JVM brede deserialisatie filter factory aangeroepen. Dit maakt de filters dynamisch en context specifiek. Zie referentie [1] voor codevoorbeeld(en).

{ PREVIEW EN INCUBATOR FEATURES }

Features die in preview zijn kunnen nog compleet wijzigen in volgende versies van Java. Het is niet te adviseren om deze in productie te gebruiken. Preview features moeten daarom speciaal geactiveerd worden. Dit doe je met de `-source 17 --enable-preview` parameters. De `-Xlint:preview` parameter is niet nodig, maar geeft output over welke preview features er in de code van je project gebruikt worden.
Incubator modules zijn een middel om niet-definitieve API's en niet-definitieve tools in handen te geven van ontwikkelaars, terwijl de API's / Tools in een toekomstige release kunnen leiden tot acceptatie of verwijdering.

406: Pattern Matching for switch (Preview)

Uitbreiding van het switch statement met patroonherkenning. In Listing 3 zijn twee voorbeelden uitgewerkt. In de `formatterPatternSwitch` method wordt gekeken naar type en in method `testPatternWithNull` wordt aangetoond dat null's nu ook onderdeel van de matching kunnen zijn. Listing 3 laat het commando en het resultaat zien.

L4

```
public class JEP406 {
    static String formatterPatternSwitch(Object o) {
        return switch (o) {
            case Integer i -> String.format("int
            %d", i);
            case Long l    -> String.format
            ("long %d", l);
            case Double d  -> String.format
            ("double %f", d);
            case String s  -> String.format
            ("String %s", s);
            default        -> o.toString();
        };
    }
    static void testPatternWithNull(String s) {
        switch (s) {
            case null -> System.out.println
            ("Oops");
            case "Java Magazine",
                  "ivonet.nl" -> System.out.
                   println("Great");
            default -> System.out.println("Ok");
        }
    }
    public static void main(String[] args) {
        System.out.println
        (formatterPatternSwitch(42));
        System.out.println
        (formatterPatternSwitch(42L));
        System.out.println
        (formatterPatternSwitch(42D));
        System.out.println
        (formatterPatternSwitch("42"));
        System.out.println
        (formatterPatternSwitch(false));
        testPatternWithNull(null);
        testPatternWithNull("Java Magazine");
        testPatternWithNull("ivonet.nl");
        testPatternWithNull("other");
    }
}
```

14

```
$ java --enable-preview --source 17 JEP406.java
Note: JEP406.java uses preview features of Java
SE 17.
Note: Recompile with -Xlint:preview for details.
int 42
long 42
double 42.000000
String 42
false
Oops
Great
Great
Ok
```

412: Foreign Function & Memory API (Incubator)
Dit is de voortzetting van het werk dat is begonnen met JEP 370 in Java SE 14 en JEP 383 voor toegang tot Foreign Memory en JEP 389 Foreign Linker API. Het biedt een manier om te werken aan geheugen buiten de Java-runtime (dus buiten de garbagecollector) en biedt een alternatief voor Java Native Interface (JNI). Dit item is uitgebreid behandeld in het artikel over Java 14 [2].

414: Vector API (Second Incubator)
Deze JEP introduceert een API om vector berekeningen te compileren tot optimale vector instructies op de ondersteunde CPU-architecturen. Dit zorgt voor een verbeterde implementatie en performance. Deze JEP bouwt voort op JEP 338 die in Java 16 is geïntroduceerd. Zie referentie [2] voor voorbeeldcode.

Deprecated en verwijderde Features
Deze sectie beschrijft JEP's die gemarkeerd worden als deprecated of verwijderd zijn. Deprecated betekent dat ze gevlagd zijn voor toekomstige verwijdering.

398: Deprecate the Applet API for Removal
In Java SE 9 was de Applet API al deprecated verklaard, maar nog niet gemarkeerd om verwijderd te worden. Nu wel. Aangezien alle webbrowsers deze support al lang verwijderd hebben, is het irrelevant om deze API nog te laten bestaan.

407: Remove RMI Activation
RMI Activation is een verouderd gedeelte van RMI (Remote Method Invocation). De in Java SE 15 deprecated verklaarde (JEP 385) RMI Activation is in Java SE 17 verwijderd.

410: Remove the Experimental AOT and JIT Compiler
De experimentele AOT- en JIT-compilers die de Graal-compiler gebruiken, worden verwijderd. De experimentele features worden nagenoeg niet gebruikt. Het onderhouden ervan kost veel tijd en moeite. Als je deze features toch wilt gebruiken, kun je GraalVM gebruiken.

411: Deprecate the Security Manager for Removal
De Security Manager stamt uit Java SE 1.0, maar is al jaren niet meer de primaire methode om client-side code te beveiligen. In deze JEP wordt de Security Manager dus deprecated verklaard en zal waarschijnlijk tegelijk met de Applet API in een toekomstige versie van Java worden verwijderd.

306: Restore Always-Strict Floating-Point Semantics
Deze JEP zorgt ervoor dat alle floating-point operaties weer 'strict' zullen zijn (strictfp), waardoor resultaten over alle platformen weer hetzelfde zullen zijn. Eigenlijk gaan we met deze JEP een terug in de tijd naar de periode voor Java SE 1.2. Eind jaren 90 waren er wat eigenaardigheden in de x87 floating-point co-processor instructie set van de populaire x86 architectuur. Dat in combinatie met een niet helemaal soepel lopende interactie tussen Java en de JVM van die tijd waren de redenen om hier aanpassingen in te maken. Met de huidige processoren en evolutie van de JVM is dit niet meer nodig, waardoor strict weer de default kan zijn.

{ CONCLUSIE }

Je kunt gerust zeggeen dat sinds de vorige LTS versie (11), Java een hele evolutie heeft doorgemaakt. Vroeger (pre Java SE 9) kwam er ongeveer elke drie jaar een grote versie van Java uit en was het dus lang wachten om te doorgronden wat er allemaal met de taal gebeurd was. Het half jaarlijks releasen heeft dit proces gelukkig een stuk transparanter gemaakt. De nieuwe manier van releasen maakt het een stuk makkelijker en minder foutgevoelig om te upgraden en het stelt ontwikkelaars in staat om in een vroeg stadium te wennen aan geïmplementeerde nieuwe features. <

{ REFERENCES }

1 https://openjdk.java.net/projects/jdk/17/
2 http://ivo2u.nl/oz
3 https://github.com/openjdk