Java 12 bis 17- Neue Features

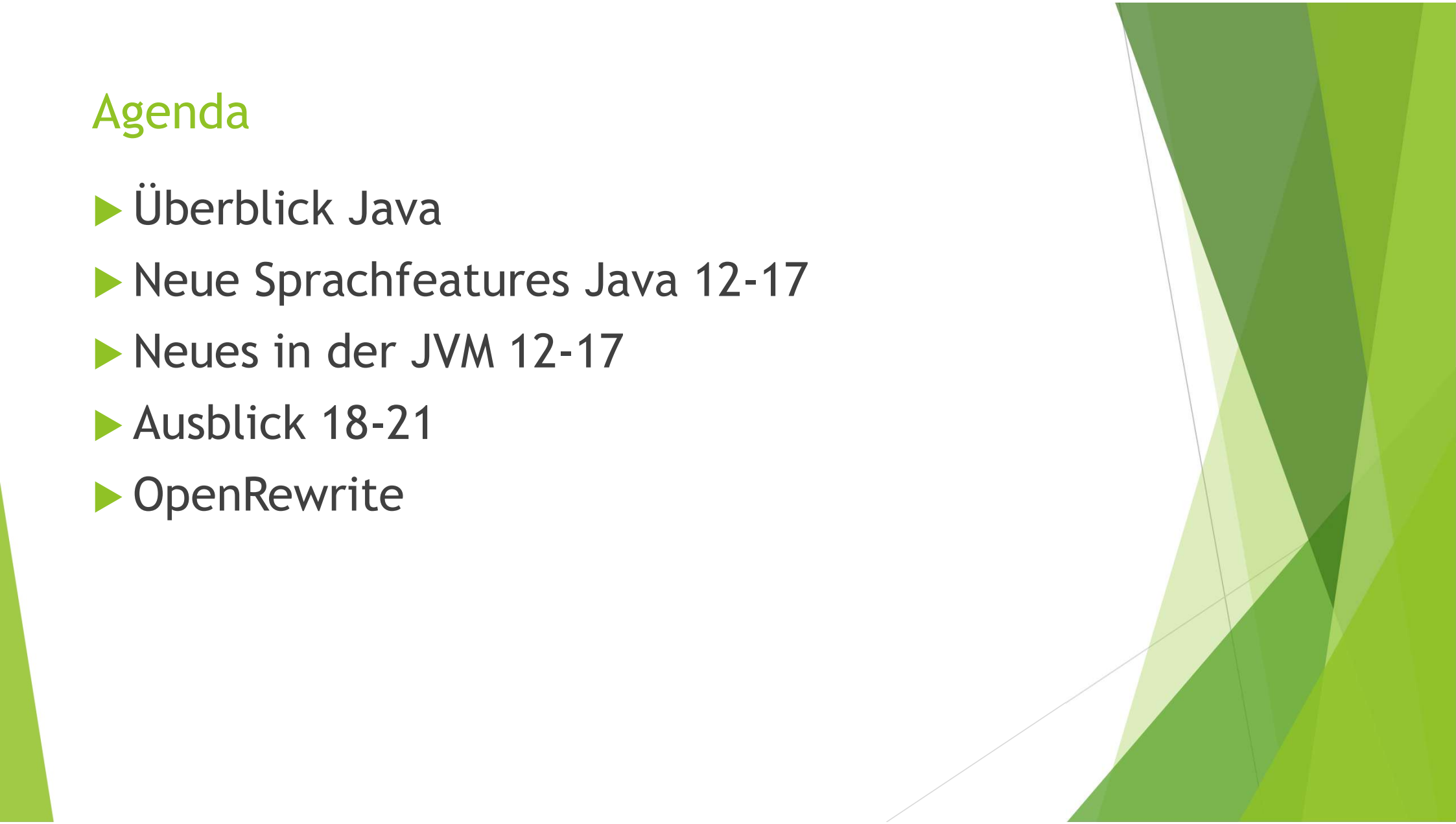
Vorstellung

- Trainer: Michael Zöller
- Aus Hamburg
- Software Architect, developer and trainer
- Xing: https://www.xing.com/profile/Michael_Zoeller3
- LinkedIn: https://www.linkedin.com/in/michael-z%C3%B6ller-579041256
- freelancermap : https://www.freelancermap.de/profil/michael-zoeller
- Mail: michael2.zoeller@gmail.com

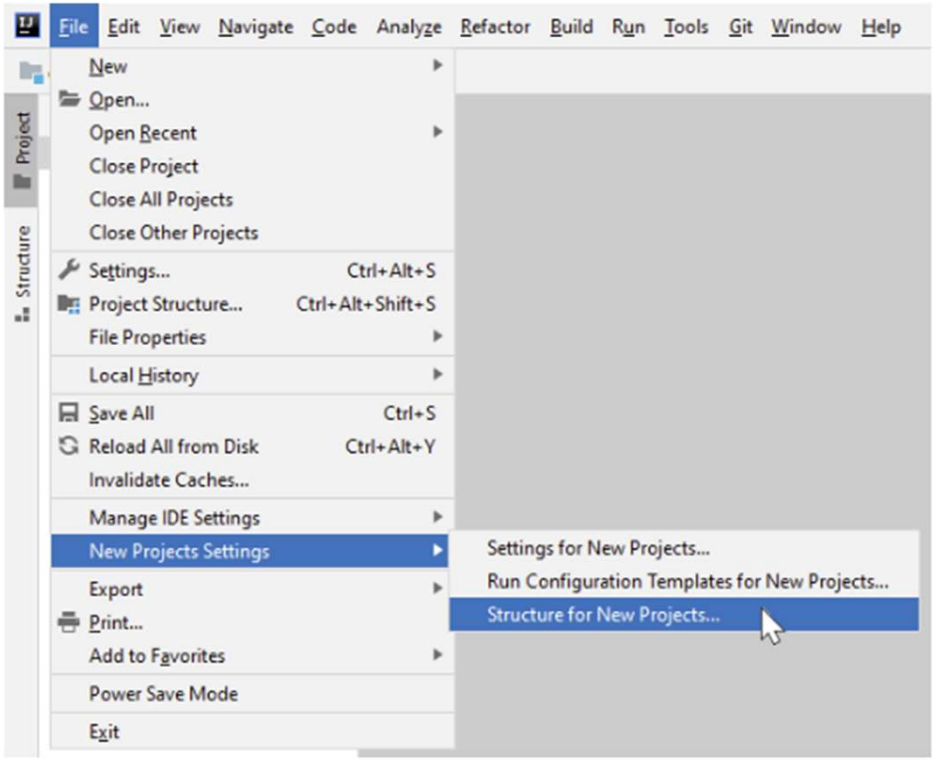
Du oder Sie?

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-17
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite



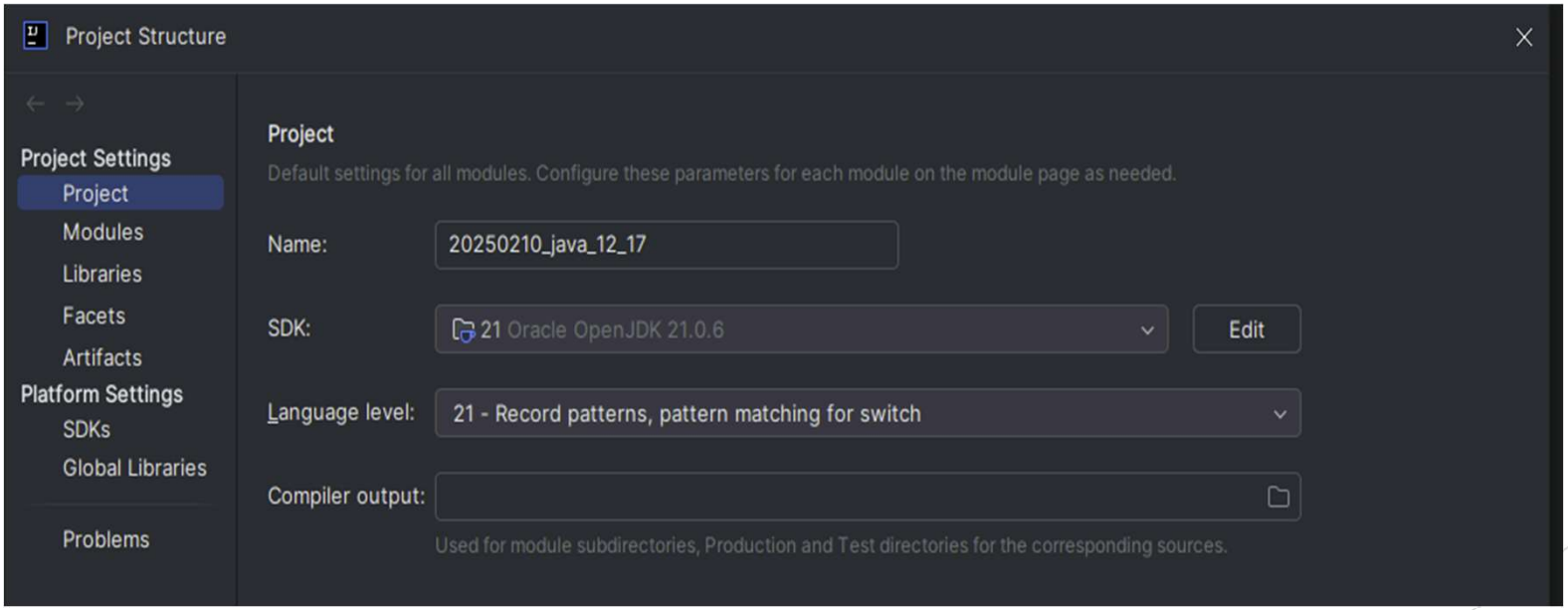
IntelliJ – default sdk für Projekte auswählen



- Dort jdk 21 einstellen

Repository

- https://github.com/MichaelZett/20250210_java_12_17



Agenda

- Überblick Java
  - **Java historisch**
  - Der Java Releaseprozess
  - Aktuelle Projekte
- Neue Sprachfeatures Java 12-17
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite

Java Historie – Write Once, run everywhere

- Erste stabile Java Version (1.0.2) erschien im Januar 1996
- Java setzte sich von Anfang an als **die** neue Sprache durch:
  - **Plattformunabhängigkeit**: Java wurde mit dem Prinzip "Write Once, Run Anywhere" (WORA) entworfen, was bedeutet, dass Java-Programme, die auf einer Plattform kompiliert wurden, auf jeder anderen Plattform ausgeführt werden können, die über eine Java Virtual Machine (JVM) verfügt. Diese Eigenschaft machte Java besonders attraktiv für das Internet und Unternehmensanwendungen.
  - **Objektorientierte Programmierung (OOP)**: Java ist eine größtenteils objektorientierte Sprache, was in den 1990er Jahren der jüngste Hype war.
  - **Robustheit und Sicherheit**: Z.B. Verbot von direktem Zugriff auf Speicheradressen, um gängige Programmierfehler und Sicherheitsrisiken zu vermeiden (keine Pointer-Arithmetik).
  - **Automatische Speicherverwaltung**: Java führt eine automatische Speicherverwaltung durch (Garbage Collection), die hilft, Speicherlecks und andere Speicherprobleme zu vermeiden.

Java Historie – Weitere Vorzüge Javas

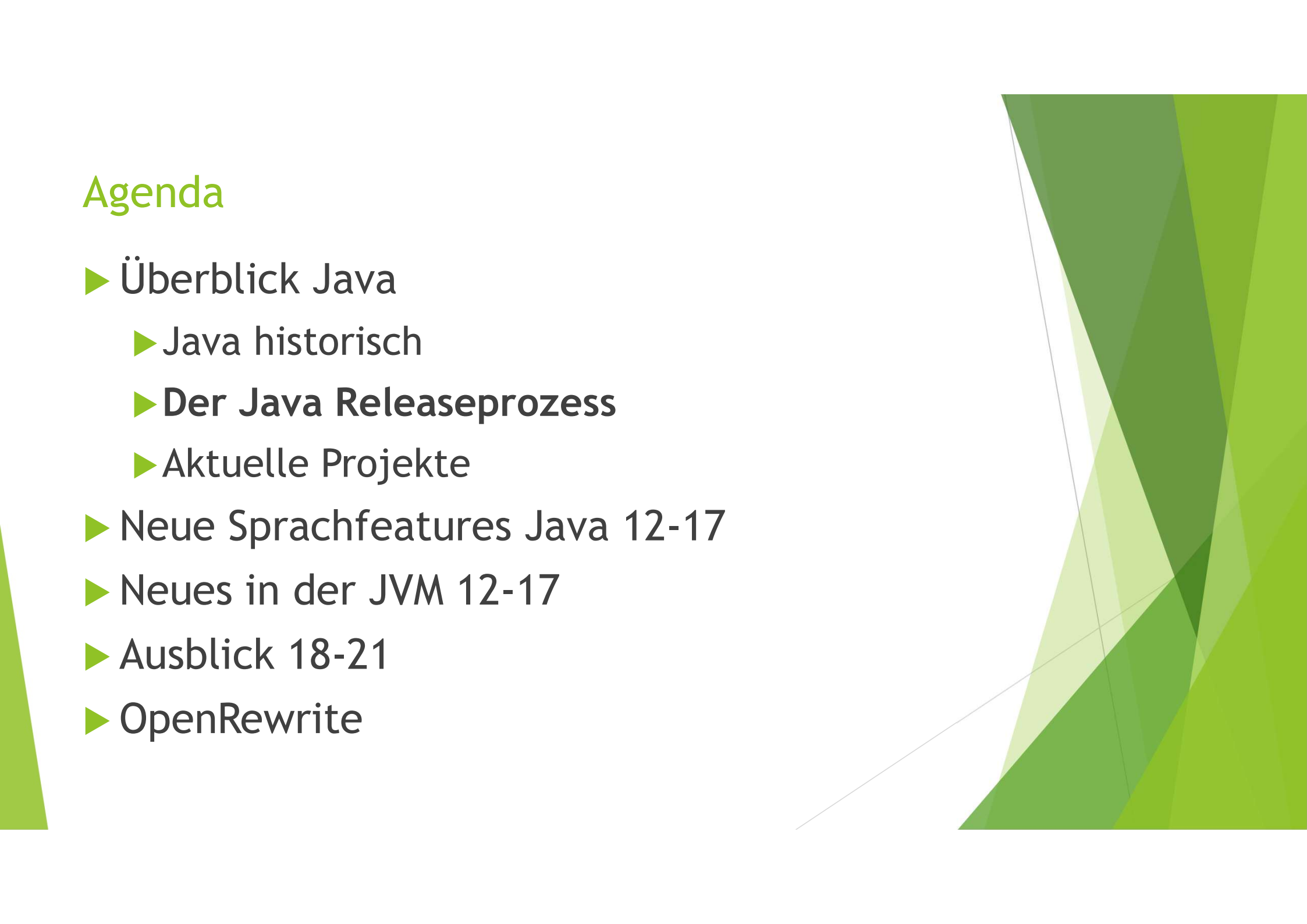
- **Reiche Standardbibliotheken**: Java bot sehr früh eine umfangreiche Sammlung von Standardbibliotheken an: Netzwerkprogrammierung, Dateizugriff, Benutzeroberflächengestaltung und Datenbankverbindung.
- **Einfachere Einstiegshürde für Entwickler**: Eine zu C und C++ vergleichbare Syntax, den damals vorherrschenden Sprachen für Unternehmensanwendungen. Es fiel diesen Entwicklern leichter, Java zu lernen und effektiv in der neuen Sprache zu programmieren. Java das bessere, weil sichere und einfachere C++.
- **Anpassungsfähigkeit und Skalierbarkeit**: Java arbeitete kontinuierlich an seiner Hauptschwäche Performance und erwies sich als äußerst anpassungsfähig an neue Technologietrends (Internet, Mobile, IoT, Funktional, Cloud).

Java Code im Laufe der Zeit

- „Opa, wie hast Du eigentlich damals Java entwickelt?“
  - Text Editor, wenn man Glück hatte mit Syntax-Highlighting
  - Kompiliert mit javac im Terminal
  - IDE – wir hatten doch nichts!
  - Jetzt zeig‘ ich Dir mal Code von mir….
    - Projekt 01_history

Agenda

- Überblick Java
  - Java historisch
  - **Der Java Releaseprozess**
  - Aktuelle Projekte
- Neue Sprachfeatures Java 12-17
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite



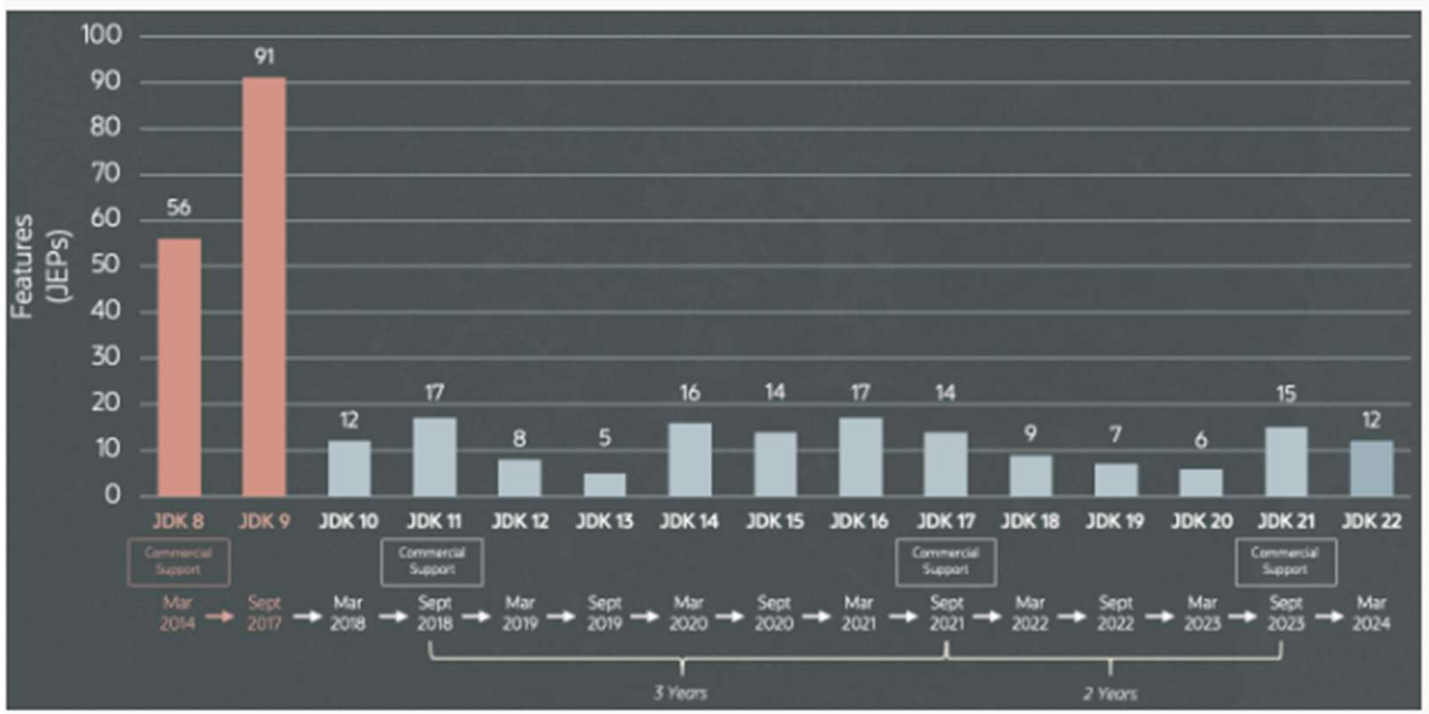
Java Entwicklungsprozess

- JCP: Java Community Process - Der JCP formalisiert die Art und Weise, wie neue Funktionen und Änderungen an Java (d.h. technische Spezifikationen) vorgeschlagen, geprüft und genehmigt werden, einschließlich der Definition verschiedener Rollen, die Personen einnehmen können.
- JSR: Java Specification Request – Vorschlagsprozess für “reifere” feature requests (Scheitern nicht erwartet)
- JEP: JDK Enhancement Proposal – leichtgewichtigerer Prozess für feature requests (Scheitern möglich)
- https://openjdk.org/jeps/0 - Liste aller JEPs

Neues „Feature Releases“ Modell seit Java 10

- Java Releases sollen sich nicht mehr verzögern (wie Java 7-9)
- Sie erscheinen seit Java 10 immer halbjährlich im März und September
- Major-Versionen erhalten nun lediglich ein halbes Jahr Patches bis zum nächsten Major (außer LTS Versionen)
- Alle 3 Jahre (**ab Java 21 alle 2 Jahre**) wird eine Version zur **L**ong **T**erm **S**upported Version erkoren, die für 8 Jahre LTS Security Patches erhält
  - Java 8 wurde auch zur LTS befördert mit Patches bis 2030
- Features, die noch in der Entwicklung sind und Feedback brauchen, sind dabei - aber müssen speziell eingeschaltet werden (Incubator, Preview)
- Es hat sich schnell gezeigt, dass dieses Vorgehen vorteilhaft ist, da neue Features so direkt produktions-gehärtet werden

Vorhersehbarkeit



Unterschiedliche Reifegrade

1. Preview: für neue Java platform features, die vollständig spezifiziert und implementiert, aber noch offen für Änderungen sind
2. Experimental: vor allem für neue features in der JVM
3. Incubating: für potentielle neue APIs und JDK tools

Entwicklung über mehrere Versionen

	Java 10	Java 11	Java 12	Java 13	Java 14	Java 15 (*)
Local-Variable Type Inference - `var`	Standard	→	→	→	→	→
Local-Variable Syntax for Lambda Parameters		Standard	→	→	→	→
Switch Expressions			Preview	2nd Preview	Standard	→
Text Blocks				Preview	2nd Preview	Standard
Records					Preview	2nd Preview
Pattern Matching for `instanceof`					Preview	2nd Preview
Sealed Classes						Preview

Lizenzmodelländerung Java 11

- Ab Java 11 wurde die zuvor übliche „Oracle Binary Code License“ (BCL) für neue Oracle-JDK-Releases durch eine Lizenz ersetzt, die kostenlose Nutzungen für geschäftliche Zwecke stark einschränkte.
- Während Oracle das Oracle JDK weiterhin zum Download bereitstellte, durfte man im kommerziellen Umfeld nur kostenlose Updates bis zum sogenannten „Ende des öffentlichen Updates“-Zeitpunkt nutzen. Danach benötigte man ein kostenpflichtiges Abonnement (Java SE Subscription), um weiterhin Sicherheitsupdates und Support von Oracle zu erhalten.
- Gleichzeitig gab (und gibt) es aber das *OpenJDK* (offiziell von Oracle und anderen), das weiterhin unter der *GPLv2 mit Classpath Exception*-Lizenz steht. Außerdem existieren diverse andere Distributoren wie z. B. Eclipse Temurin (Adoptium), Amazon Corretto, Red Hat, Azul, BellSoft, usw., die kostenlose und langfristig aktualisierte Builds anbieten.

Lizenzmodelländerung Java 17

- „Oracle No-Fee Terms and Conditions“ (NFTC) - Mit Java 17 (einem „Long-Term-Support“-Release) hat Oracle eine neue Lizenz eingeführt: Für die Laufzeit dieser Version (sprich bis zum Ende der Updates für Java 17) können die meisten Nutzer das Oracle JDK wieder frei einsetzen – auch kommerziell/produktiv.
- Die NFTC-Lizenz räumt eine kostenfreie Nutzung ein, allerdings nur für eine gewisse Zeit. Für Java 17 ist diese im September 2024 geendet und kommerzielle Nutzer benötigen ein kostenpflichtiges Abo von Oracle.
- Die aktuell kostenlose LTS ist Java 21
- Ausführliche Details stehen in den „No-Fee Terms and Conditions“.

Lizenzmodell - Auf Nummer sicher gehen

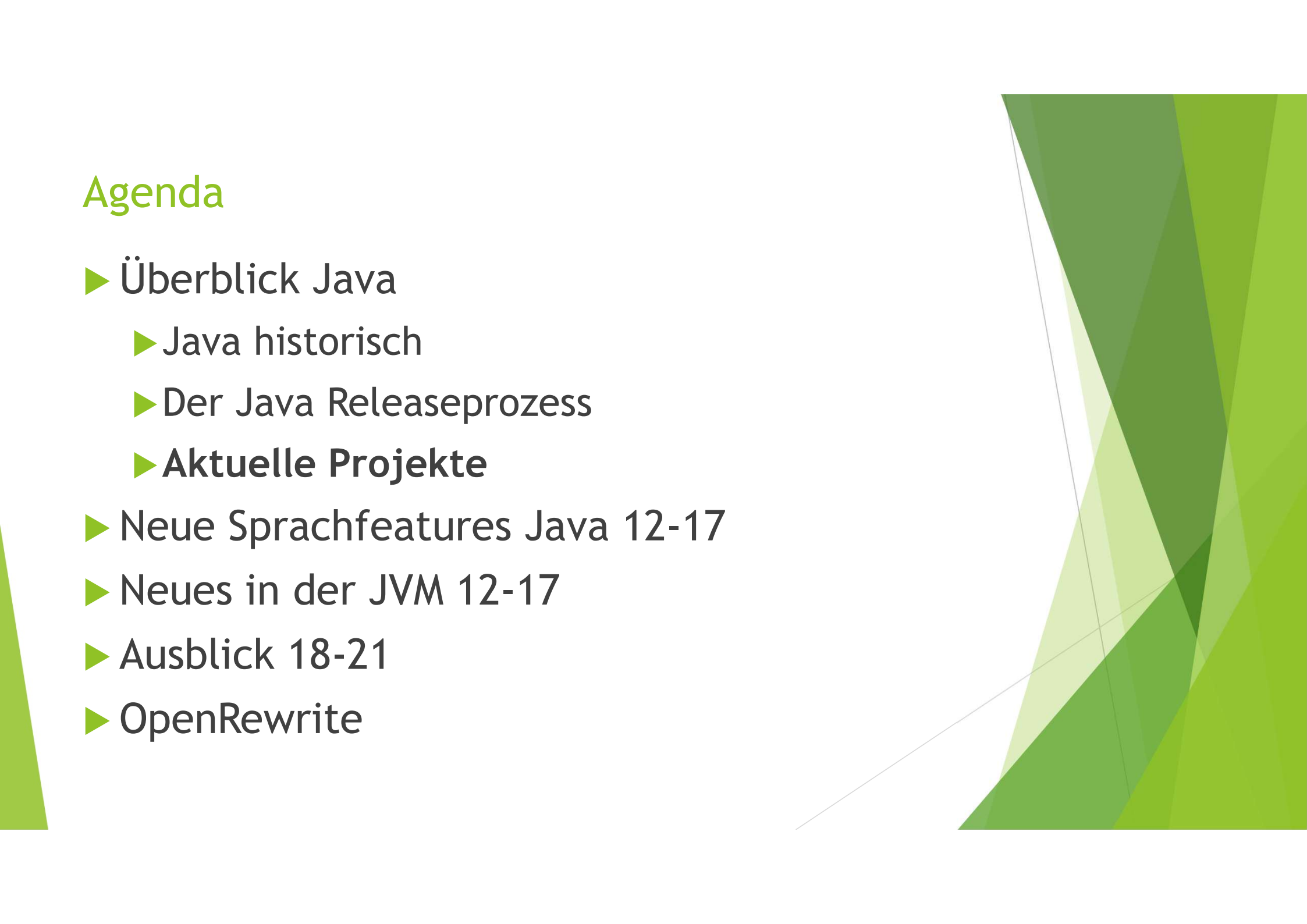
OpenJDK-Distributionen anderer Anbieter sind weiterhin eine gute Option, weil sie kostenfrei und oft mit Langzeit-Patches angeboten werden. Beispiele:

- Eclipse Temurin (Adoptium)
- Amazon Corretto
- Azul Zulu
- BellSoft Liberica

- Diese Distributionen sind vollkommen kostenfrei, auch für kommerzielle Nutzung, und bieten ebenfalls Sicherheitsupdates über den gesamten LTS-Zeitraum.

Agenda

- Überblick Java
  - Java historisch
  - Der Java Releaseprozess
  - **Aktuelle Projekte**
- Neue Sprachfeatures Java 12-17
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite



Projekte

- Projekte und Roadmap: https://openjdk.org/jeps/1
- Amber – Kleine Verbesserungen für die Produktivität
- Babylon – Code Reflection, GPU Programming
- Liliput – Footprint reduzieren durch Kompression der object header
- Loom – Virtual Threads, Structured Concurrency
- Panama – Foreign Memory Access, Vektorrechnung
- Valhalla – Value Objects
- ZGC - a scalable low-latency garbage collector

Project Amber

- Das Ziel von Project Amber ist es, kleinere, produktivitätsorientierte Java-Sprachfeatures zu erforschen und zu entwickeln.
- Im Rahmen von Project Amber wurde z.B. auch local type inference (var) entwickelt, das größte Themenfeld ist/war aber „Pattern Matching“
- Pattern Matching in Java ist bislang mit Regex und Strings assoziiert, in anderen (funktionalen) Programmiersprachen definiert man es aber als:
- „Einen Mechanismus zur Überprüfung eines Wertes anhand eines Musters. Eine erfolgreiche Übereinstimmung kann einen Wert auch in seine Bestandteile zerlegen.“
- Die Switch Expressions (Java 14) sind der 1. Baustein, Pattern Matching für instanceof (16) der nächste.
- Weiter geht es mit: Records (16), Sealed Classes (17), Pattern Matching für Switch (21), Record Pattern (21) und mehr

Hintergründe und neue Themen

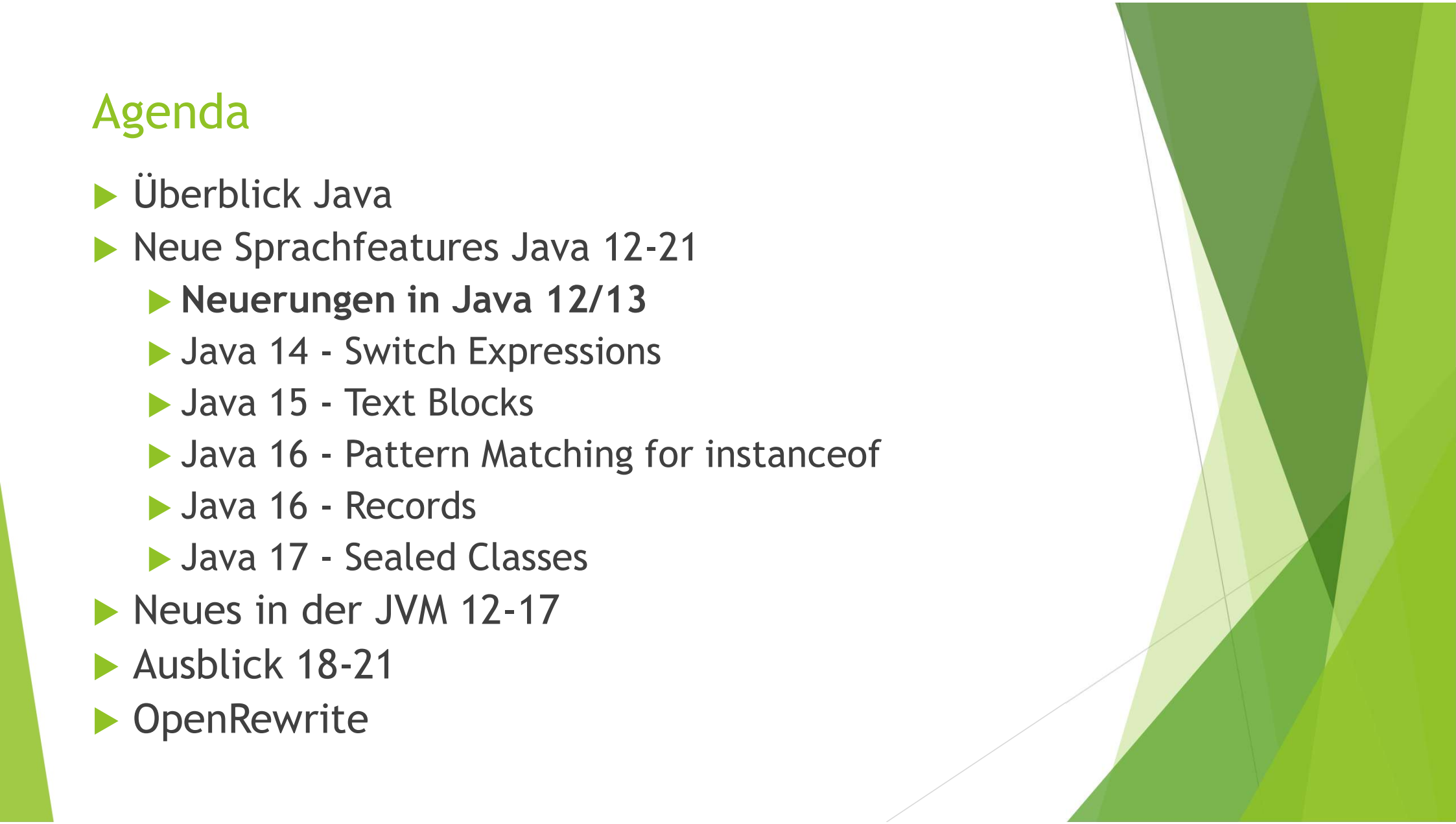
- https://www.infoq.com/articles/java-sealed-classes/
  - Data-centric Programming: Algebraische Typen nutzen und Pattern Matching einsetzen
  - Unterstütze Szenarien, besonders in gut verstandenen Domänen, wo Datenkapselung nicht genug Vorteile bringt sondern sogar Einfachheit und Transparenz erschwert
- Aktuelle Themen:
  - Primitive types in Patterns, instanceof, and switch
  - Statements before super(…)
  - String Templates
  - Implicitly Declared Classes and Instance main Methods

Projekt Valhalla

- Das Projekt Valhalla erweitert das Java-Objektmodell um Value Objects sowie Primitive Classes.
- Hier werden die Abstraktionen der objektorientierten Programmierung mit den Leistungsmerkmalen einfacher Primitiver Typen kombiniert:
  - *“Codes like a class, works like an int.”*
- Vorarbeiten: Nest-Based Access Control, Dynamic Class-File Constants (delivered in 11), JVM Constants API (12), Hidden Classes (15) und Warnings for Value-Based Classes (16)
- Nächstes Ziel: https://openjdk.org/jeps/401: Value Classes and Objects (Preview) – noch kein Zieltermin
- Dann noch: primitive classes, migrating the existing primitives, and universal generics

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-21
  - **Neuerungen in Java 12/13**
  - Java 14 - Switch Expressions
  - Java 15 - Text Blocks
  - Java 16 - Pattern Matching for instanceof
  - Java 16 - Records
  - Java 17 - Sealed Classes
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite



Java 12 – 19.03.2019

1. New Language Features

- Compact Number Formatting
- Teeing Collector
- File::mismatch Method
- New String methods indent(),transform()

2. JVM Improvements

- G1GC Improvements
- Default CDS Archives

3. New Tools and Libraries

- Microbenchmark Suite

4. Miscellaneous

- Diverse Security Updates und Housekeeping
- Unicode 11 support

5. Incubator and Preview Features

- JEP 325: Switch Expressions (Preview)
- ZGC Concurrent Class Unloading (experimental)
- Shenandoah: A Low-Pause-Time Garbage Collector (experimental)

Java 13 – 17.09.2019

1. New Language Features

- FileSystems.newFileSystem() Method

2. JVM Improvements

- JEP 350: Dynamic CDS Archives
- JEP 353: Reimplement the Legacy Socket API

3. New Tools and Libraries

- New keytool -showinfo -tls Command for Displaying TLS Configuration Information

4. Miscellaneous

- Diverse Security Updates und Housekeeping
- Unicode 12.1 support

5. Incubator and Preview Features

- JEP 354: Switch Expressions yield (2nd Preview)
- JEP 354: Text Blocks (Preview)
- JEP 351: ZGC: Uncommit Unused Memory (experimental)

Java12 Code

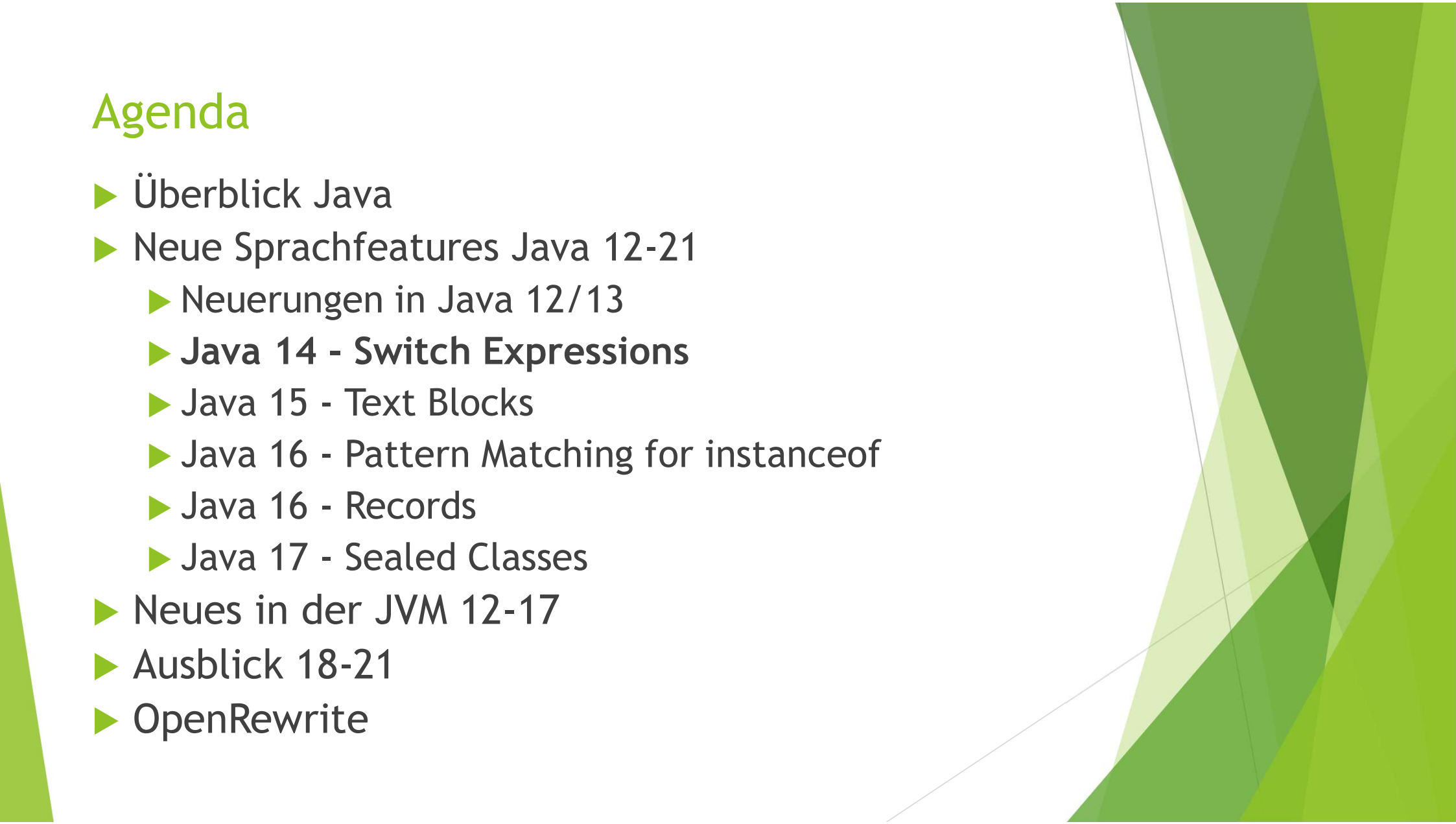
- 02_java12-17 de.zettsystems.misc

Java 12/13 – JVM Verbesserungen

- Default CDS Archives
  - Die seit Java 10 zur Verfügung stehende Option ist seit 12 default an
  - Ein solches Klassenarchiv steht allen JVM Instanzen gemeinsam zur Verfügung und reduziert daher die redundaten Haltung dieser Daten in jeder Instanz
- Dynamic CDS Archives
  - Mit Java 13 gibt es die Möglichkeit auch Archive von Anwendungsklassen dynamisch zu erzeugen und zu benutzen
  - https://docs.oracle.com/en/java/javase/17/vm/class-data-sharing.html#GUID-7EAA3411-8CF0-4D19-BD05-DF5E1780AA91
  - https://nipafx.dev/java-application-class-data-sharing/
- G1GC Verbesserungen
  - Der G1 prüft nun den Java-Heap-Speicher bei Inaktivität der Anwendung und gibt ihn ggf. an das Betriebssystem zurück.
  - Besseres Zeitmanagement durch abbrechbare Collections
- Legacy Socket API neu implementiert
  - Baut nun auf java.nio auf
  - Vorbereitung für virtual threads (Java 21)

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-21
  - Neuerungen in Java 12/13
  - **Java 14 - Switch Expressions**
  - Java 15 - Text Blocks
  - Java 16 - Pattern Matching for instanceof
  - Java 16 - Records
  - Java 17 - Sealed Classes
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite



Java 14 – 17.03.2020

1. New Language Features

- JEP 361: Switch Expressions

2. JVM Improvements

- JEP 345: NUMA-Aware Memory Allocation for G1
- Parallel GC Improvements

3. New Tools and Libraries

- JFR Event Streaming for Flight Recorder

4. Miscellaneous

- Diverse Security Updates und Housekeeping

5. Incubator and Preview Features

- JEP 368: new escapes for Text Blocks (2nd Preview)
- JEP 305: Pattern Matching for instanceof (Preview)
- JEP 359: Records (Preview)
- JEP 358: Helpful NullPointerExceptions (Preview – default aus)
- ZGC on Windows (JEP 365) and macOS (JEP 364) – Experimental
- JEP 370: Foreign Memory Access API (Incubator)
- JEP 343: Packaging Tool (Incubator)

Switch Expressions (Java 14)

- Switch wurde 1:1 von C übernommen – wahrscheinlich wie viele Entscheidungen mit der Absicht der großen Menge von C-Entwicklern Java schmackhaft zu machen
- Wegen der seltsamen Semantik (fall-through, break) wurde aber in OO-Kreisen meist if/else bevorzugt
- Denoch gab es in Java 5 (Enum und Wrapper switch-bar) und 7 (String switch-bar) Weiterentwicklungen
- Im Kontext von Aufzählungen war es schon immer die übersichtlichere Alternative
- Mit Java 14 und den Switch Expression gibt es jetzt eine funktionale Variante bei der auch OO-Enthusiasten nicht mehr die Nase rümpfen müssen
- Mit Java 21 gibt es dann sogar die nächste Weiterentwicklung

Switch Expression Code

- 02_java12-17 de.zettsystems.switchexpression

Quiz

```
enum Direction {
  UP, DOWN, HOLD;
}
…
public static void main(String[] args) {
  var dir = Direction.UP;
  switch(dir) {
    case UP -> System.out.print("up ");
    case DOWN ->  System.out.print("down"); // line n1
  }
}
```

Which statement is true? Choose one.

A. The code compiles and prints up.

The answer is A.

B. The code compiles and prints up down.

The answer is B.

C. The code fails to compile. To fix it you must add the following after line n1:
`case HOLD -> System.out.println("hold");`

The answer is C.

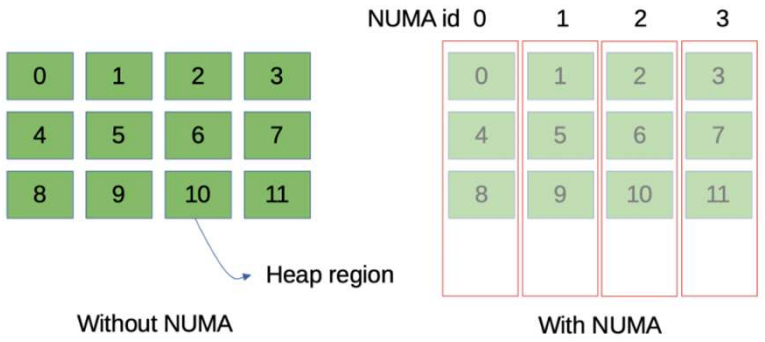
D. The code fails to compile. To fix it you must add the following after line n1:
`default -> System.out.println("default");`

The answer is D.

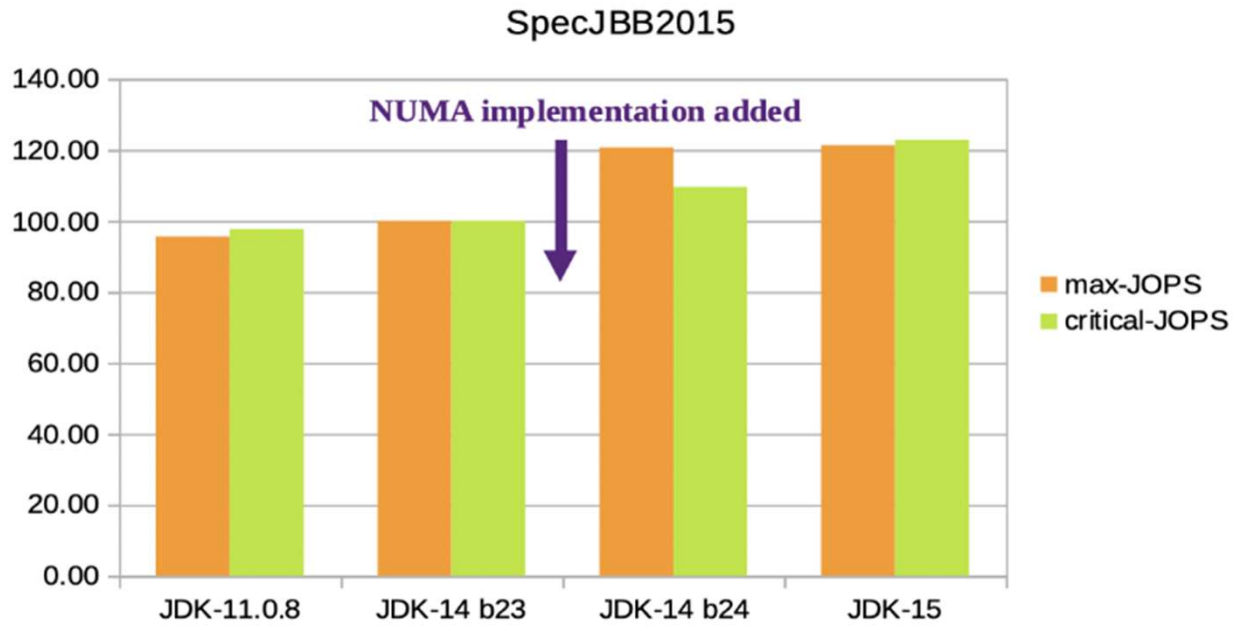
A ist richtig,
es ist Arrow Syntax aber im Switch Statement
und nicht Expression: kein fall-through,
aber man muss auch nicht alle Fälle abdecken

Java 14 – JVM Verbesserungen

- Parallel GC Improvements
  - Parallel GC hat denselben Task-Management-Mechanismus für die Planung paralleler Aufgaben übernommen wie andere GCs. Dies kann zu erheblichen Leistungsverbesserungen führen.
- NUMA-Aware Memory Allocation for G1
  - Moderne Multi-Socket-Maschinen haben zunehmend ungleichmäßigen Speicherzugriff (NUMA), d. h. der Speicher ist nicht von jedem Sockel oder Kern gleich weit entfernt. Der Parallel GC, der durch -XX:+UseParallelGC aktiviert wird, ist schon seit vielen Jahren NUMA-fähig. G1 hat dies nun auch mit -XX:+UseNUMA verfügbar



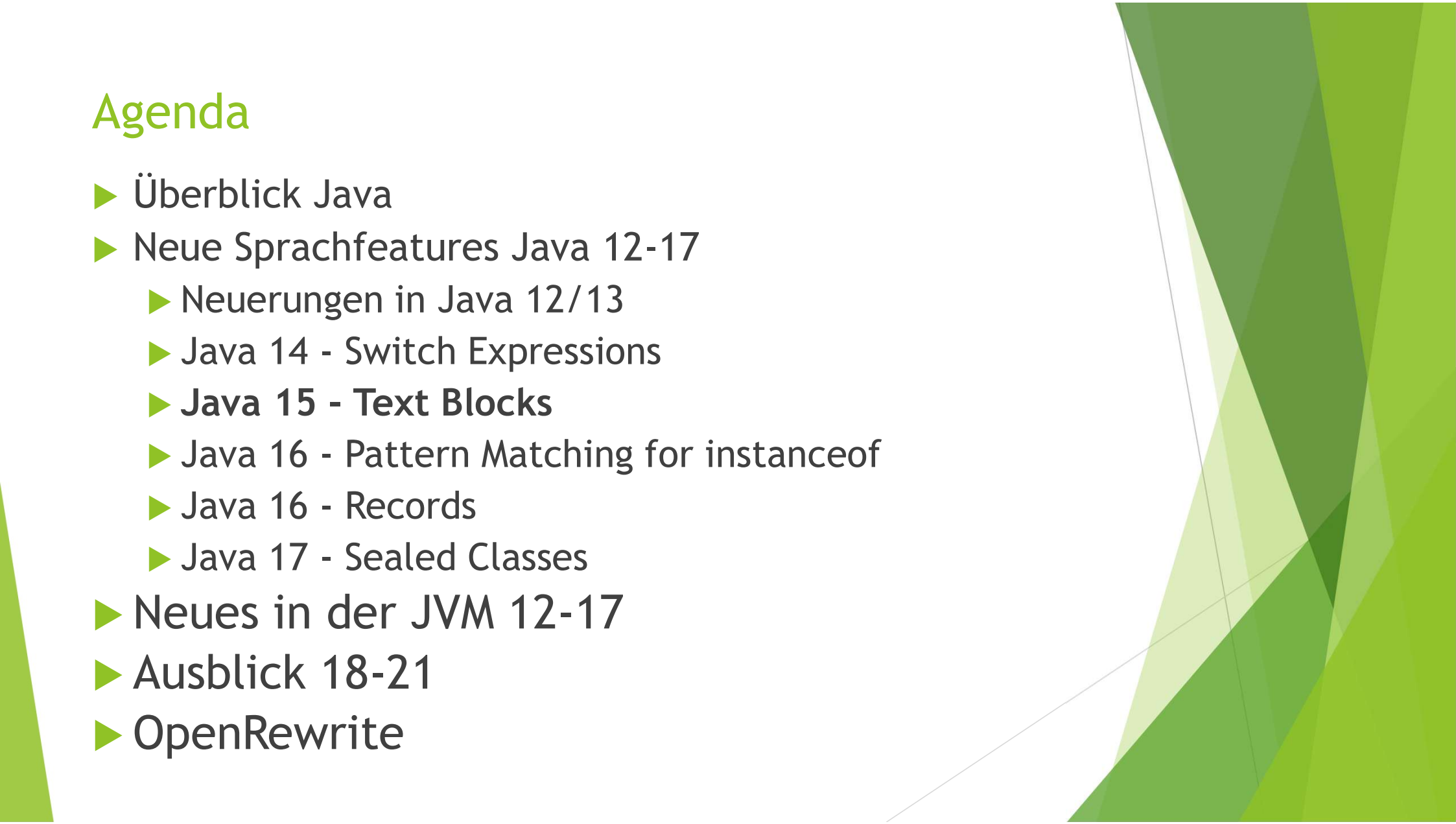
Java 14 – Numa Verbesserung



https://sangheon.github.io/2020/11/03/g1-numa.html

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-17
  - Neuerungen in Java 12/13
  - Java 14 - Switch Expressions
  - **Java 15 - Text Blocks**
  - Java 16 - Pattern Matching for instanceof
  - Java 16 - Records
  - Java 17 - Sealed Classes
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite



Java 15 – 16.09.2020

1. New Language Features

- Added isEmpty Default Method to CharSequence
- JEP 378: Text Blocks released
- JEP 371: Hidden Classes

2. JVM Improvements

- JEP 377: ZGC released
- JEP 379: Shenandoah released
- Helpful NullPointerExceptions (default an)
- DatagramSocket API reimplemented

3. New Tools and Libraries

- ---

4. Miscellaneous

- Diverse Security Updates und Housekeeping

5. Incubator and Preview Features

- JEP 384: Records (2nd Preview)
- JEP 375: Pattern Matching Type Checks (2nd Preview)
- JEP 360: Sealed Classes (Preview)
- JEP 383: Foreign Memory API (Incubator)

Text Blocks (Java 15)

- Text Blocks sind die neue Art und Weise Texte, die über mehrere Zeilen gehen, abzubilden
- Das Ergebnis eines Textblocks ist ein einfacher String
- Textblöcke beginnen mit einem """ (drei Anführungszeichen), gefolgt von optionalen Leerzeichen und einem Zeilenumbruch.
- Das Ende des Blocks ist wieder """ - was aber in der gleichen Zeile wie der letzte Text stehen kann
- Innerhalb der Textblöcke können wir Zeilenumbrüche und Anführungszeichen frei verwenden, ohne dass ein Zeilenumbruch erforderlich ist.
- So können wir wörtliche Fragmente von HTML, JSON, SQL oder was auch immer wir brauchen, auf elegantere und lesbarere Weise einfügen.
- Im resultierenden String werden der (Basis-)Einzug und der erste Zeilenumbruch nicht berücksichtigt.
- Dazu neue Methode formatted() in String als Alternative für String.format()…

Text Blocks und NPE Code

- 02_java12-17 de.zettsystems.textblocks und de.zettsystems.misc

Quiz

The closing delimiter of a Java text block is:

Select 1 option(s):

- """
- """ immediately followed by a new line
- """ followed by any number of white spaces and then a new line
- """ followed by one or more white spaces
- A new line followed by """.

A ist richtig

Quiz 2

Given:

```
String s1 = "Hello World";
String s2 = """
            Hello World""";
String s3 = """
            Hello World
            """;
System.out.println((s1 == s2)+" "+s2.equals(s3)+" "+s2.intern().equals(s3.intern()));
```

What is the result?

Select 1 option(s):

- ○ `true false true`
- ○ `true true true`
- ○ `false true false`
- ○ `true false false`
- ○ `false false false`

D (true false false) ist richtig

Quiz 3

What will the following code print?

```
String s1 = """
            a \
            b \t
            c \s
            """;
System.out.println(s1.length()+" "+s1.split("\\n").length);
```

Select 1 option(s):

- 11 3
- 12 3
- 11 4
- 10 2
- 10 3
- 9 2

E isr richtig (10 2)
s1 contains: "a b \t\nc \s\n" i.e. a total of 10 characters.

Finally, this string is being split using the new line character. String's split method returns an array of Strings. But the split method does not include trailing empty strings in the resulting array. That is why, although the string pointed to by s1 ends with a new line and thus contains 3 lines, the split method returns only 2 strings - "a b \t" and "c \s". Thus, s1.split("\\n").length returns 2.

Java 15 – JVM Verbesserungen

- 2 neue GCs released
  - Shenandoah (nicht bei oracle dabei) und ZGC
  - Beide zielen auf Server mit großen Speicher (Terrabyte) und niedrige Latenzzeiten ab (kaum “stop-the-world”)
  - `java -XX:+UseZGC`
- Helpful NullPointerExceptions released
  - Es gibt mehr Infos, was denn genau null war
- DatagramSocket API re-implementiert
  - Nach socket-api nun diese 2. API, die auf virtual threads (Java 21) vorbereitet wurde

Hidden classes

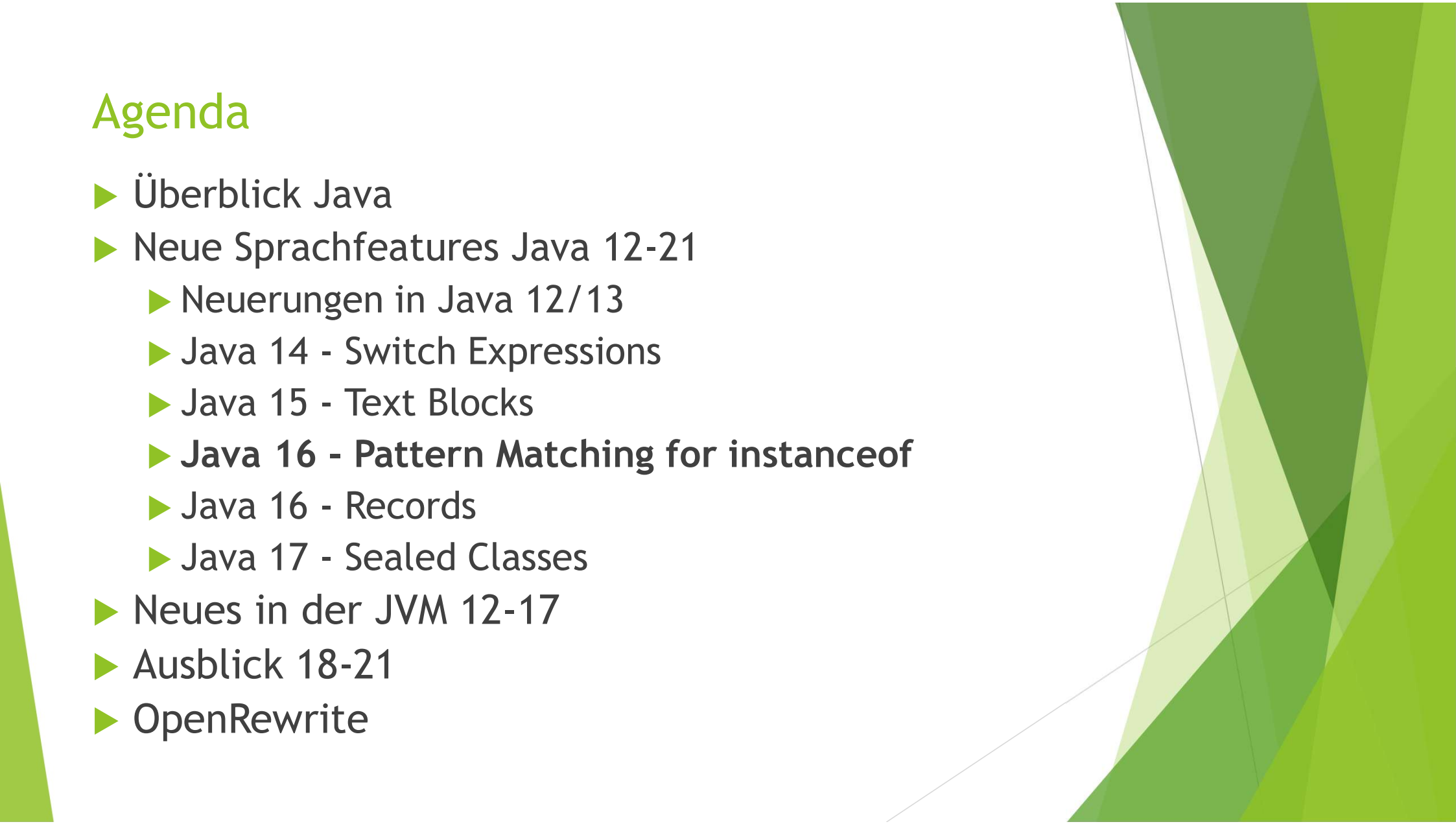
- Hidden classes
  - Können nicht direkt vom bytecode anderer Klassen genutzt werden
  - Können nicht zum Deklarieren von Feldern, als Parameter, Return Wert oder Superclass benutzt werden
  - Können nicht durch classloader via Class::forName, ClassLoader::loadClass gefunden warden
  - Der Lebenszyklus kann feiner gesteuert werden als bei Anonymen Klassen
- Ein feature vor allem für Sprach/Framework/Tool-Entwickler
- Beispielsweise erzeugte der Compiler in Java für einen Lambda Ausdruck bytecode, um dynamische eine anonyme Klasse zu erzeugen – jetzt wird dafür eine hidden class genutzt
- Löst teilweise „sun.misc.Unsafe::defineAnonymousClass“ ab (und soll möglichst erweitert werden, um das ganz abzulösen)

Aufgaben

- Bitte alle TODOs in 02_java_12_17 package de.zettsystems.exercises12_15 lösen

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-21
  - Neuerungen in Java 12/13
  - Java 14 - Switch Expressions
  - Java 15 - Text Blocks
  - **Java 16 - Pattern Matching for instanceof**
  - Java 16 - Records
  - Java 17 - Sealed Classes
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite



Java 16 – 16.03.2021

1. New Language Features

- JEP 394 Pattern Matching for instanceof
- JEP 395 Records
- Add Stream.toList() Method

2. JVM Improvements

- JEP 376 ZGC Concurrent Thread Processing
- JEP 387 Elastic Metaspace
- Concurrently Uncommit Memory in G1

3. New Tools and Libraries

- JEP 380 Unix-Domain Socket Channels
- JEP 392 Packaging Tool jpackage

4. Miscellaneous

- JEP 390 Warning for Value-Based Classes
- JEP 396 Strongly Encapsulate JDK Internals by default (flag)

5. Incubator and Preview Features

- JEP 338 Vector API (Incubator)
- JEP 389 Foreign Linker API (Incubator)
- JEP 393 Foreign Memory Access API (3rd Incubator)
- JEP 397 Sealed Classes (2nd Preview)

Pattern Matching inctanceof (Java 16)

- Bei der Typprüfung mit instanceof können wir nun direct eine Variable des entsprechenden Typs initialisieren
- Das spart den vorher üblichen Cast, der üblicherweise immer erfolgte
- Diese Variable können wir darüber hinaus direkt weiter in der if-Anweisung verwenden
- Aber ACHTUNG! – hier wird “Flow Scope” verwendet:
  - eine Pattern Matching-Variable ist nur dann bekannt, wenn der instanceof-Test bestanden wurde – nach der if-Anweisung, z.B. in einem else-Zweig ist sie nicht bekannt

Pattern Matching inctanceof Code

- 02_java12-17 de.zettsystems.pminstanceof

Quiz

Given:

```
public class TestClass
{
  public static void main(String args[])
  {
    B b = new C();
    A a = b;
    if (a instanceof B b1) b1.b();
    if (a instanceof C c1) c1.c();
    if (a instanceof D d1) d1.d();
  }
}
class A {
    void a(){ System.out.println("a"); }
}
class B extends A {
    void b(){ System.out.println("b"); }
}
class C extends B {
    void c(){ System.out.println("c"); }
}
class D extends C {
    void d(){ System.out.println("d"); }
}
```

What is the result?

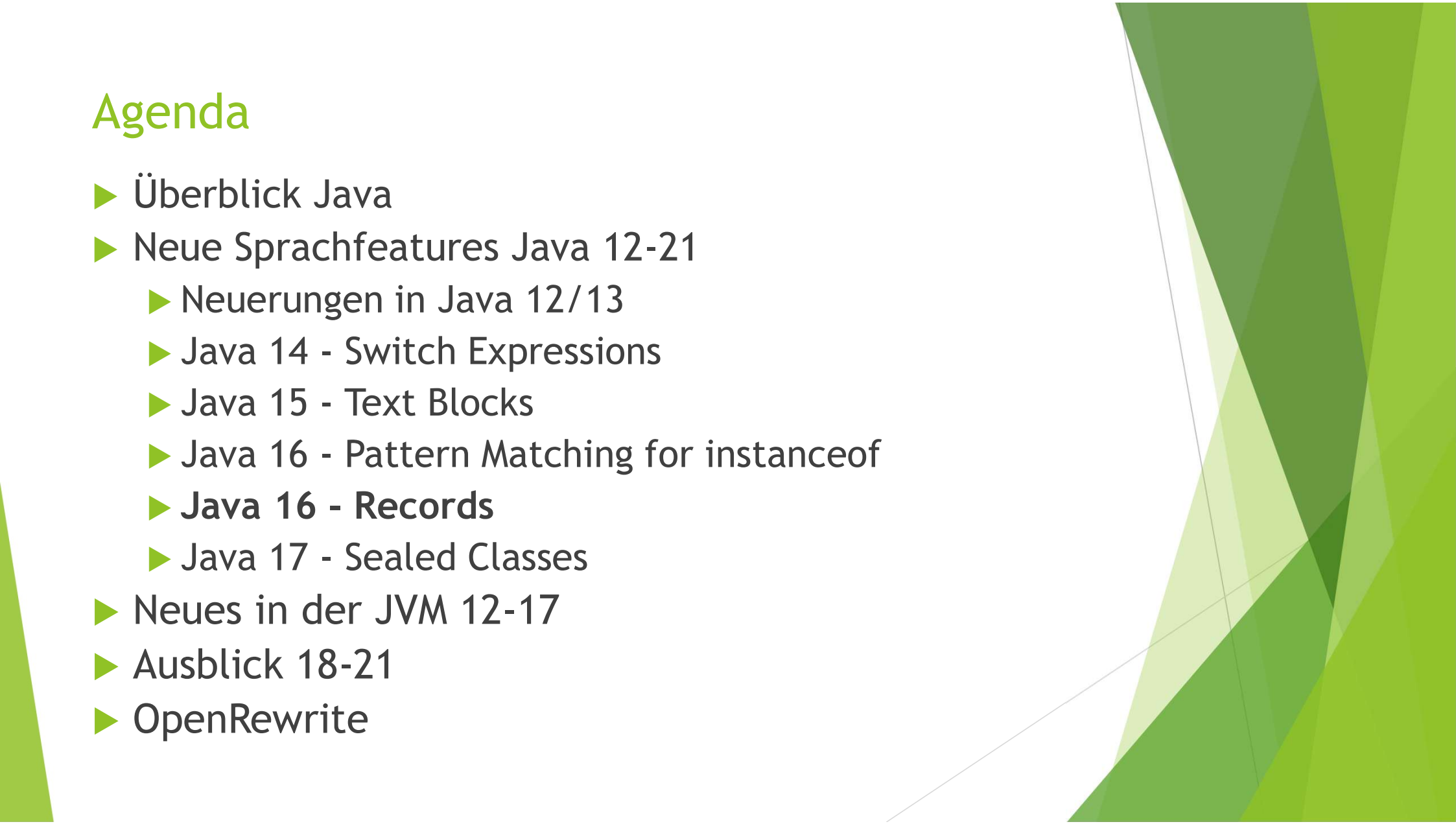
Select 1 option(s):

- Compilation failure.
- ClassCastException thrown at run time.
- a
  b
- b
  c
- b
  c
  d

A ist richtig

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-21
  - Neuerungen in Java 12/13
  - Java 14 - Switch Expressions
  - Java 15 - Text Blocks
  - Java 16 - Pattern Matching for instanceof
  - **Java 16 - Records**
  - Java 17 - Sealed Classes
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite



Records (Java 16)

- Ein neuer Grundtyp in Java (Neben Klasse, Interface, Annotation und Enum), um immutable Objekte zu erzeugen
- Immutable – Gültigkeit der Daten ohne synchronized sichergestellt:
  - private final field für jedes Datum
  - Getter für jedes Feld, aber kein Setter
  - öffentlicher Konstruktor mit einem entsprechenden Argument für jedes Feld
- Bisher mit recht viel Boilerplate oder etwa durch Lombok @Value herstellbar
- Die „getter“ von Records heißen wie die Attribute, also ohne get/is
- Man kann weitere Konstruktoren mit unterschiedlichen Parameterlisten ergänzen
- Man kann den Standard-Konstruktor (compact constructor) anpassen – das ist gedacht, um Validierungen einzufügen
- Statische Variablen und Methoden sind genauso möglich wie in classes
- Keine Vererbung, record implizit final
- … aber Implementierung von Interfaces möglich

Records und Stream.toList() Code

- 02_java12-17 de.zettsystems.record und de.zettsystems.misc

Quiz

Identify correct statement(s) about Java records.

Select 2 option(s):

- [ ] Records are implicitly final.
- [ ] Records are implicitly static.
- [ ] Records are meant to restrict the number of instances of that type.
- [ ] Records are implicitly sealed.
- [ ] Record instances are immutable.

A,E sind richtig

Quiz 2

Given:

```
public record Student(int id) {
}
```

What is inserted by the compiler automatically in this record?

Select 3 option(s):

- [ ] Canonical constructor
- [ ] A `toString` method.
- [ ] A `clone` method.
- [ ] An `equals` method.
- [ ] A `getId` method.
- [ ] A zero-argument/no-args constructor.

A, B, D sind richtig

Java 16 – JVM Verbesserungen

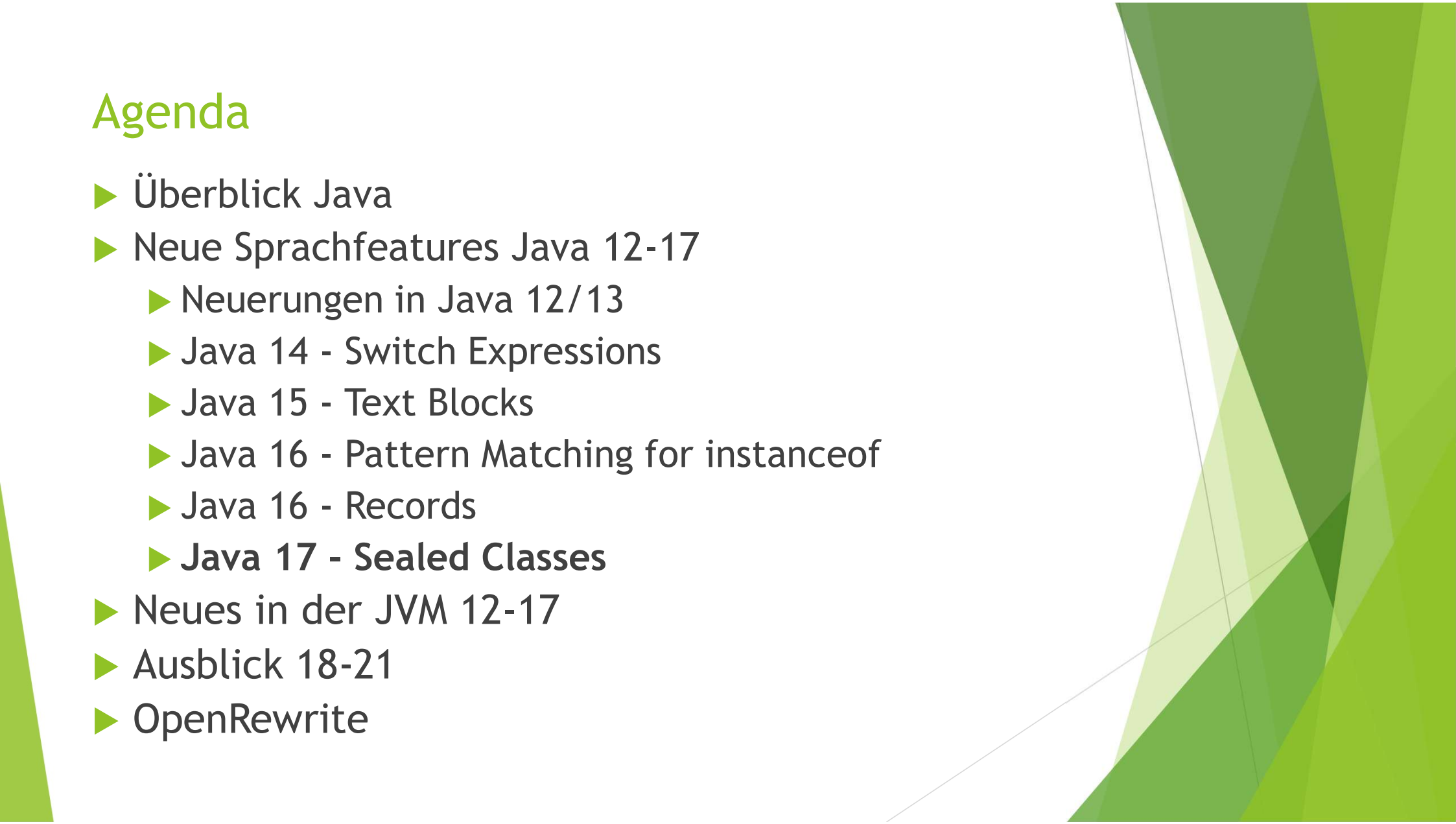
- JEP 376 ZGC Concurrent Thread Processing
  - Um stop-the-world weiter zu reduzieren (< 0.1ms) mussten weitere Prozessschritte in nebenläufige Phasen verschoben werden
- JEP 387 Elastic Metaspace
  - Rückgabe von ungenutztem Metaspace-Speicher an das OS, Verringerung des Metaspace-Footprints und Vereinfachung des Metaspace-Codes, um die Wartungskosten zu senken.
- Concurrently Uncommit Memory in G1
  - Diese neue Funktion ist immer aktiviert und ändert den Zeitpunkt, zu dem G1 Java-Heap-Speicher an das Betriebssystem zurückgibt. G1 trifft während der GC-Pause weiterhin Größenentscheidungen, verlagert aber die teure Arbeit auf einen Thread, der gleichzeitig mit der Java-Anwendung läuft

(Warning for) Value-Based Classes

- Das Projekt valhalla hat langfristig das Ziel „value objects“ einzuführen, das sind immutable Objekte ohne Identität
- Kandidaten dafür sind: die Wrapper-Klassen, Optional, Date/Time, collection implementation usw.
- Man möchte also die Performance von den primitiven Typen mit „richtigen“ Objekten kombinieren
- Nach einigen Vorarbeiten tief in der JVM wurden in diesem JEP nun die Kandidaten als @ValueObjects annotiert sowie deren public-Konstruktoren deprecated
- Neben der deprecation-Warning gibt es eine weitere Warning, wenn man diese Klassen unsachgemäß in synchronize verwendet

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-17
  - Neuerungen in Java 12/13
  - Java 14 - Switch Expressions
  - Java 15 - Text Blocks
  - Java 16 - Pattern Matching for instanceof
  - Java 16 - Records
  - **Java 17 - Sealed Classes**
- Neues in der JVM 12-17
- Ausblick 18-21
- OpenRewrite



Java 17 – 14.09.2021 (LTS bis 2029)

1. New Language Features

- JEP 409: Sealed Classes released

2. JVM Improvements

- ---

3. New Tools and Libraries

- ---

4. Miscellaneous

- Diverse Security Updates und Housekeeping
- JEP 306: Restore Always-Strict Floating-Point Semantics
- JEP 356: Enhanced Pseudo-Random Number Generators
- JEP 403: Strongly Encapsulate JDK Internals (flag weg)

5. Incubator and Preview Features

- JEP 406: Pattern Matching for switch (Preview)
- JEP 412: Foreign Function & Memory API (Incubator)
- JEP 414: Vector API (Second Incubator)

Sealed Classes (Java 17)

- Sealed classes sind wie z. B. Enums zum Erfassen von Alternativen in Domänenmodellen geeignet
- Sie ermöglichen es Programmierern und Compilern, Schlussfolgerungen über die Abzählbarkeit/Vollständigkeit zu ziehen
- Arbeiten mit Records und Pattern Matching zusammen, um eine stärker datenzentrierte Programmierung zu unterstützen
- Sealed classes erlauben die Vererbung explizit
- Erben selber müssen im gleichen package liegen (oder gleichem Modul) und auch wieder sealed, final oder explizit non-sealed sein
- Interfaces können auch sealed sein
- Records können auch sealed interfaces implementieren

Sealed Classes 2

- Sealed classes sind weiterhin eine Möglichkeit, die Erweiterbarkeit von Superklassen einzuschränken, ohne dafür auch die Benutzbarkeit einzuschränken
- Bislang konnten wir:
  - Klasse public final machen -> keine Erweiterbarkeit möglich
  - Klasse package-private machen -> keine Benutzbarkeit außerhalb des Pakets, Erweiterbarkeit im gleichen Paket
- Mit sealed Klassen können wir nun eine Liste von erlaubten Subklassen festlegen
  - Erweiterbarkeit ist möglich wie designed, Benutzbarkeit der super class ist nicht eingeschränkt
- Dieser Aspekt ist für framework/library Entwickler interessant

Sealed Classes Code

- 02_java12-17 de.zettsystems.sealed

Quiz

Given the following code:

```
//in file A.java
package p1;
public sealed class A permits p2.B{
}

//in file B.java
package p2;
public final class B extends p1.A{
}
```

Identify correct statements.

Select 1 option(s):

A ist richtig

- ○ The code will compile fine if both A and B are part of the same named module.
- ○ Both the classes must belong to the same package for the code to compile irrespective of whether they belong to the same module or not.
- ○ Both the classes will compile as long as they are packaged in the same jar.
- ○ Class B will compile if, instead of final, it is made non-sealed.

Quiz 2

Given:

```
interface Identifier{
    int id();
}

sealed class Person
    permits Student  //LINE A
{
}

record Student(int id,
        String subject)

        extends Person //LINE B

        implements Identifier //LINE C
{
   public static final long serialVersionUID  = 1L; //LINE D

   private String name = "unknown"; //LINE E

   String name(){ return "unknown"; } //LINE F

   public int id(){ return id; } //LINE G
}
```

Which lines will cause compilation issues?

Select 3 option(s):

- ☐ LINE A
- ☐ LINE B
- ☐ LINE C
- ☐ LINE D
- ☐ LINE E
- ☐ LINE F
- ☐ LINE G

A, B, E sind richtig

Verschiedenes

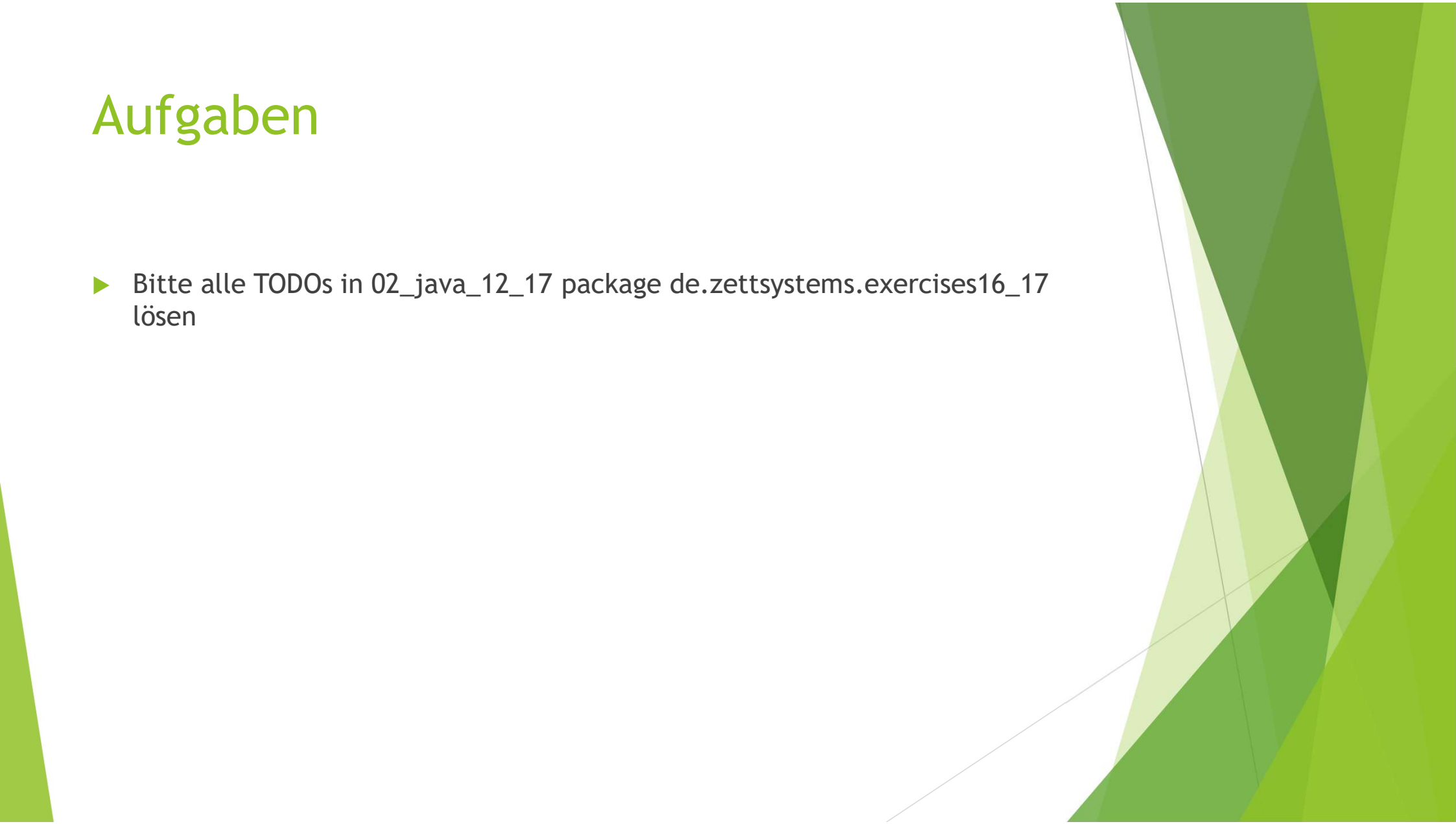
- Restore Always-Strict Floating-Point Semantics:
  - In Java 2 (1998) wurde von strict-floating-point zu einem weniger strikten Verfahren gewechselt, weil die damaligen x87-Koprozessoren mit der strikten Variante nicht klar kamen
  - Seit Anfang der 2000er kam aber mit dem Pentium eine neue Architektur auf den Markt, die strikt konnte
  - Jetzt hat man das Verfahren wieder auf das strikte umgestellt
- Enhanced Pseudo-Random Number Generators:
  - Refactoring der vorhandenen Random-Klassen, um duplizierten Code zu entfernen
  - Streaming von Random Numbers ermöglichen
  - Umsetzung neuer, teils auch besserer Random-Algorithmen

History revisited

- 02_java12-17 de.zettsystems.history : Können wir das Eingangsbeispiel verbessern?

Aufgaben

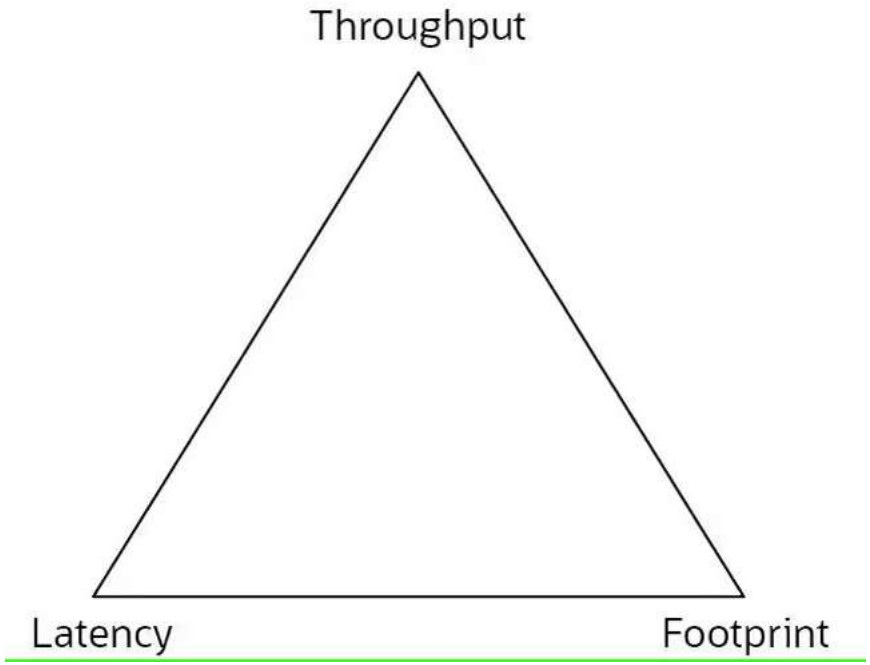
- Bitte alle TODOs in 02_java_12_17 package de.zettsystems.exercises16_17 lösen



Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-17
- Neues in der JVM 12-17
  - **Verbesserungen bei den GC**
  - Tools
- Ausblick 18-21
- OpenRewrite

GC Verbesserungen Ziele

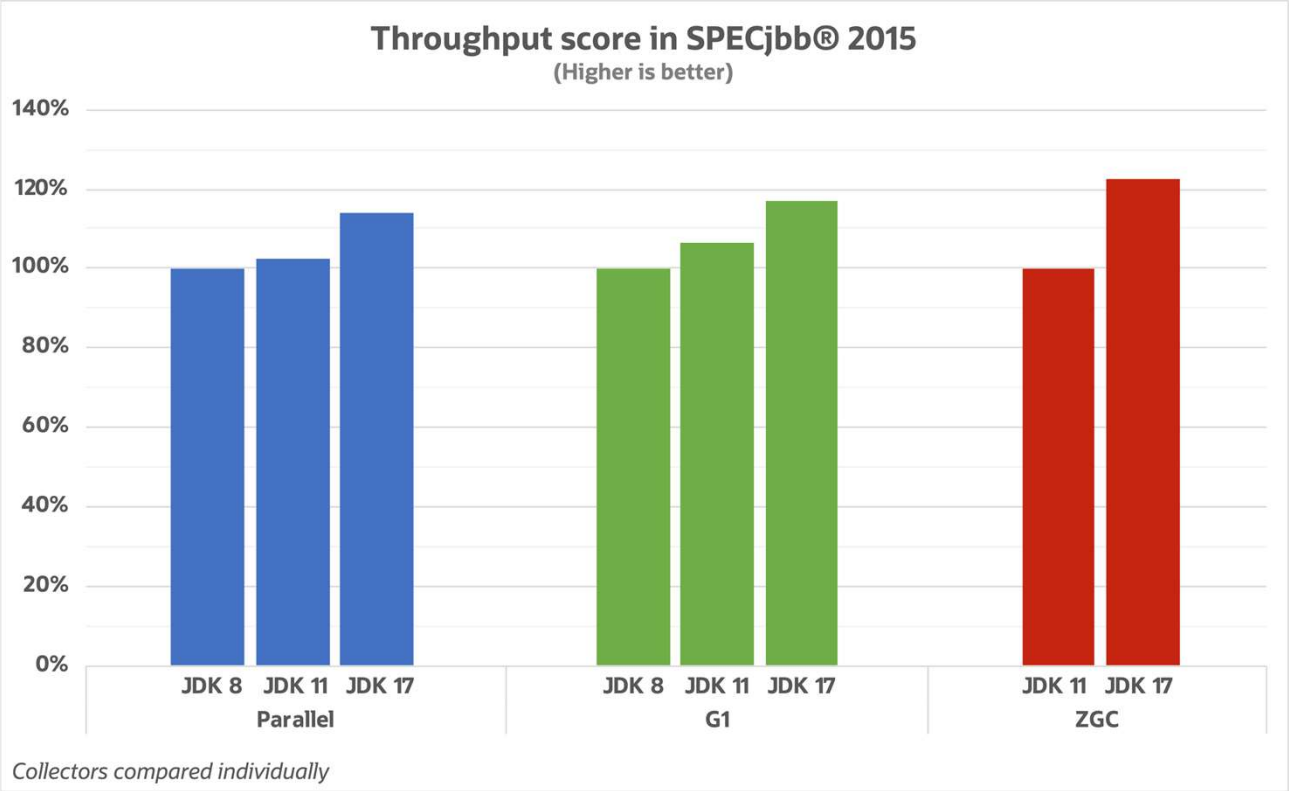


- Wir wollen hohen Throughput
- Bei niedriger Latenz und Footprint
- Unterschiedliche GC optimieren auf unterschiedliche Ziele

Relevante GCs im JDK

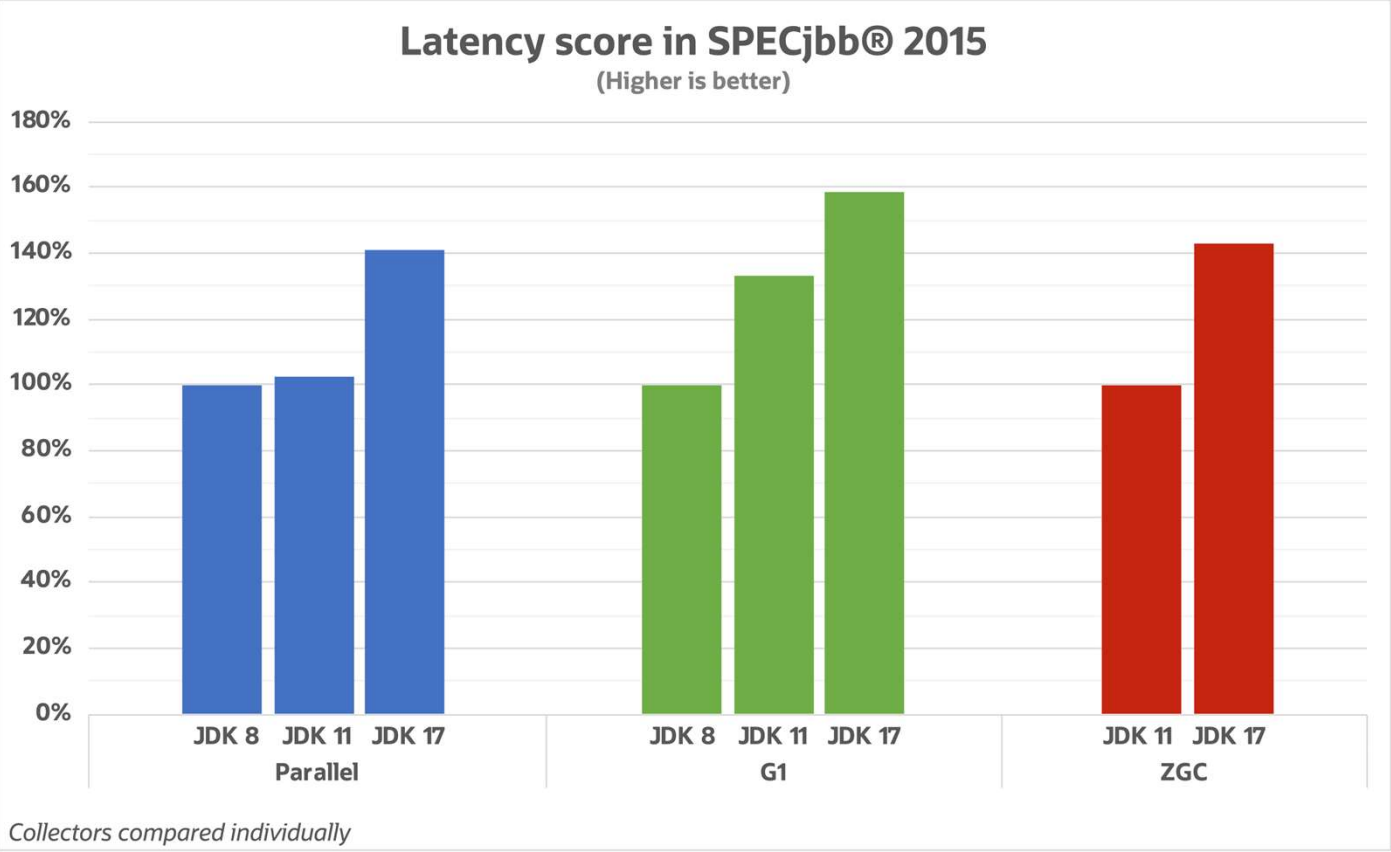
Name	Fokus	Konzept	Sonstiges
Parallel	Throughput	Multithreaded stop-the-world (STW) compaction and generational collection	Default bis JDK 8
G1	Balanced	Multithreaded STW compaction, concurrent liveness, and generational collection	Default ab JDK 9
ZGC	Latency	Everything concurrent to the application	Seit JDK 15 production-ready
Serial	Footprint and startup time	Single-threaded STW compaction and generational collection	Für Client Anwendungen

Throughput



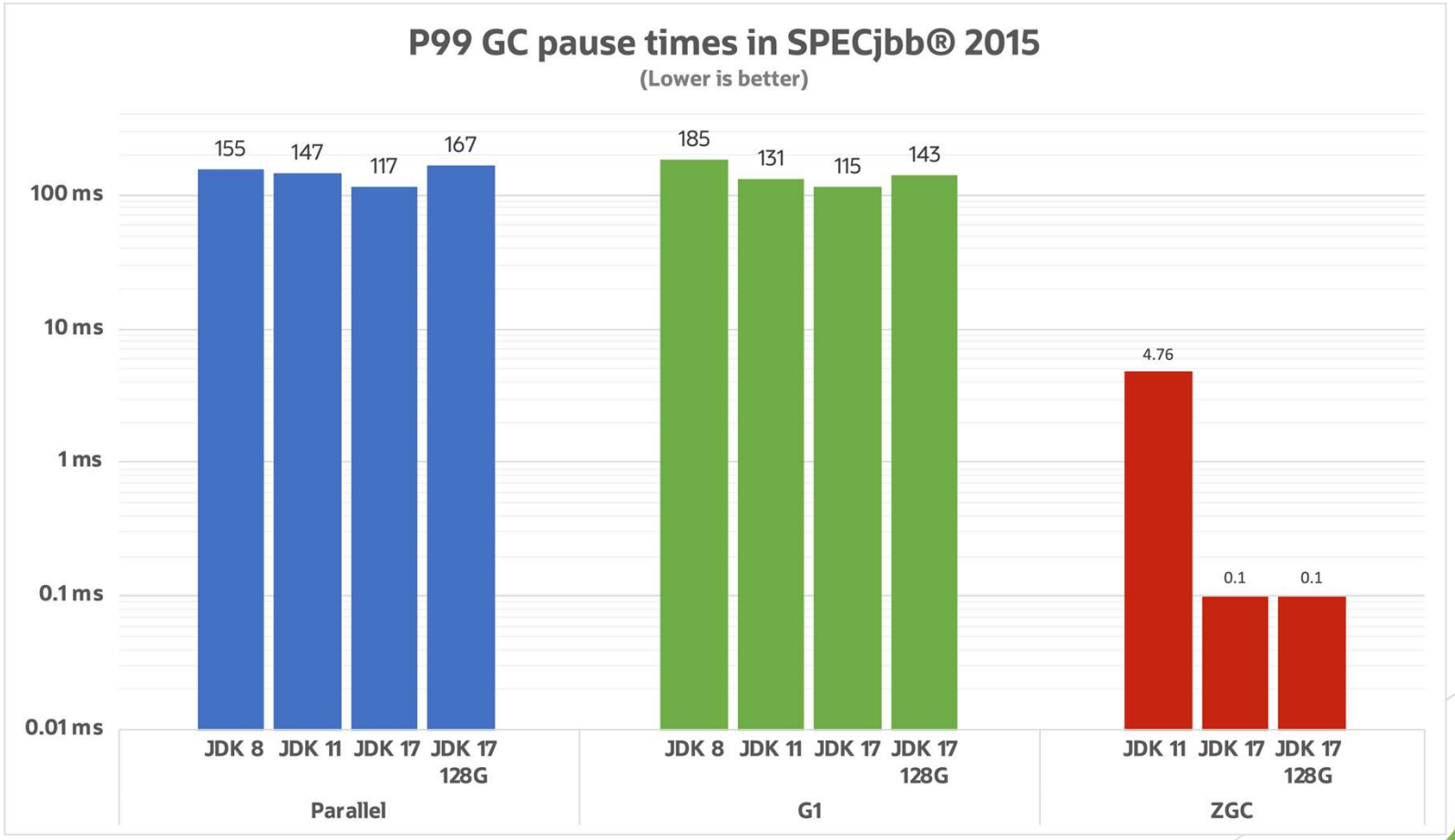
Collectors compared individually

Latency

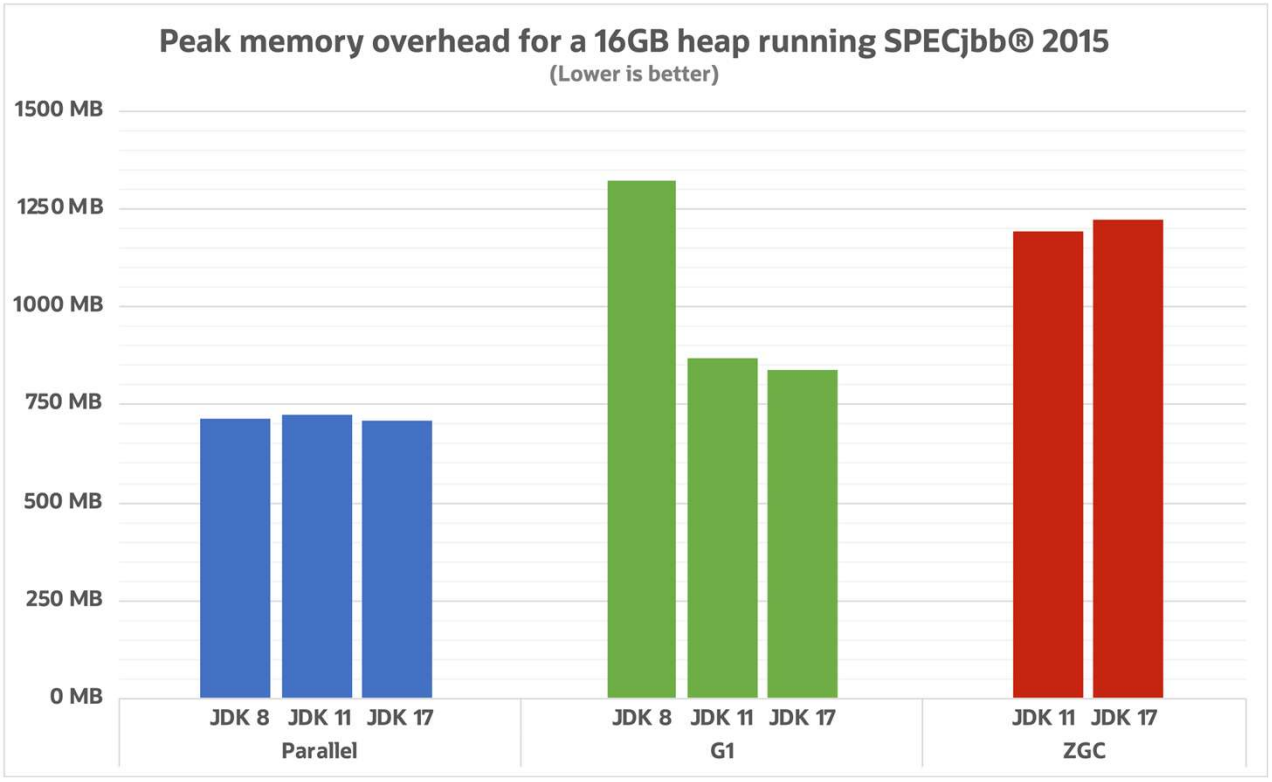


Collectors compared individually

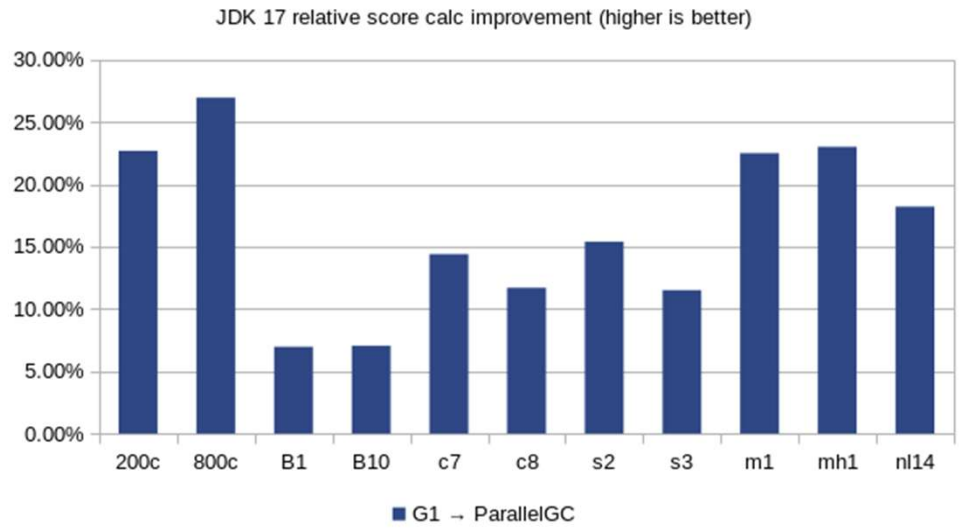
99p Latency


P99 GC pause times in SPECjbb® 2015
(Lower is better)
100 ms
10 ms
1 ms
0.1 ms
0.01 ms
155
147
117
167
185
131
115
143
4.76
0.1
0.1
JDK 8
JDK 11
JDK 17
JDK 17 128G
Parallel
JDK 8
JDK 11
JDK 17
JDK 17 128G
G1
JDK 11
JDK 17
JDK 17 128G
ZGC

Footprint


Peak memory overhead for a 16GB heap running SPECjbb® 2015
(Lower is better)
1500 MB
1250 MB
1000 MB
750 MB
500 MB
250 MB
0 MB
JDK 8
JDK 11
JDK 17
Parallel
JDK 8
JDK 11
JDK 17
G1
JDK 11
JDK 17
ZGC

Vergleich Parallel vs. G1 (optaplanner)



Executive summary

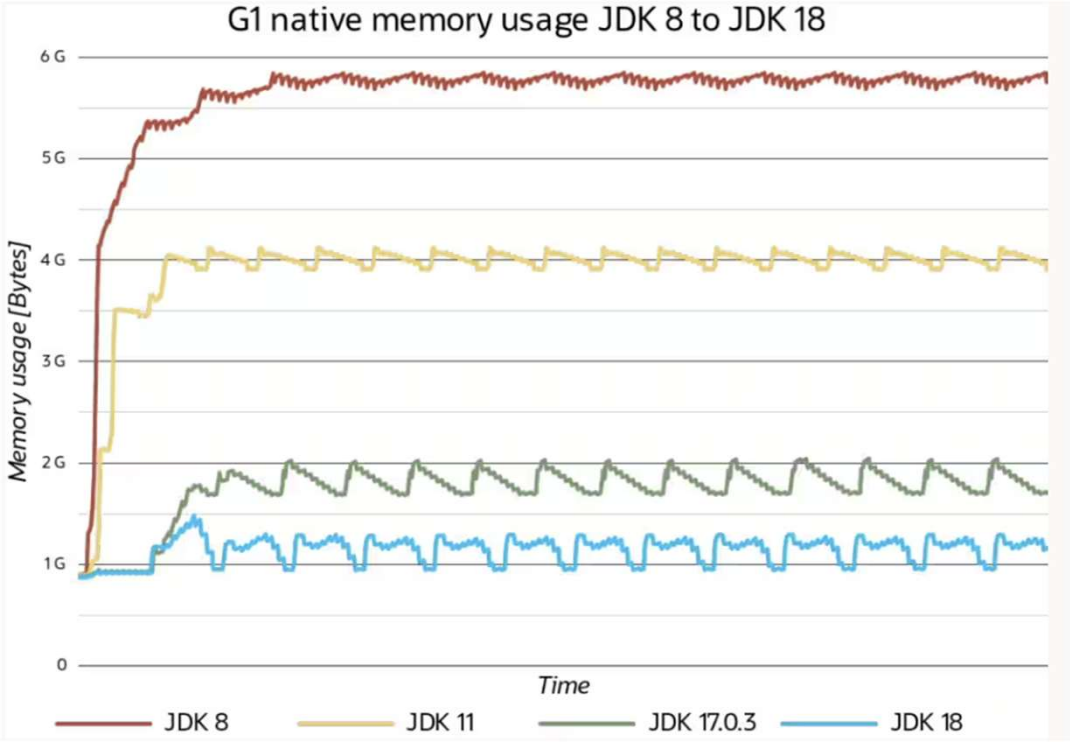
On average, for OptaPlanner use cases, these benchmarks indicate that:

- **Java 17 is 8.66% faster than Java 11** and 2.41% faster than Java 16 for G1GC (default).
- Java 17 is 6.54% faster than Java 11 and 0.37% faster than Java 16 for ParallelGC.
- The Parallel Garbage Collector is 16.39% faster than the G1 Garbage Collector.

	Average	Cloud balancing		Machine reassignment		Course scheduling		Exam scheduling		Nurse rostering		Traveling Tournament
Dataset		200c	800c	B1	B10	c7	c8	s2	s3	m1	mh1	nl14
G1GC		106,147	98,069	245,645	42,096	14,406	16,924	15,619	9,726	3,802	3,601	5,618
ParallelGC		130,215	124,498	262,753	45,058	16,479	18,904	18,023	10,845	4,658	4,430	6,641
G1 → ParallelGC	**16.39%**	22.67%	26.95%	6.96%	7.04%	14.39%	11.69%	15.39%	11.50%	22.50%	23.01%	18.20%

Table 3. Score calculation count per second on JDK 17 with different GCs

Java 18 – weitere Verbesserungen für G1



GC Verbesserungen links

- https://blogs.oracle.com/javamagazine/post/java-garbage-collectors-evolution
- https://kstefanj.github.io/2021/11/24/gc-progress-8-17.html
- https://www.optaplanner.org/blog/2021/09/15/HowMuchFasterIsJava17.html

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-17
- Neues in der JVM 12-17
  - Verbesserungen bei den GC
  - **Tools**
- Ausblick 18-21
- OpenRewrite

Microbenchmark

- Als neues Feature in Java 12 geführt, muss man aber tatsächlich extra dependencies benutzen wenn man nicht openjdk benutzt
- https://github.com/openjdk/jmh
- Tutorial: https://jenkov.com/tutorials/java-performance/jmh.html
- Kann man nutzen, um „verdächtige“ Codestrukturen auf Performance-Probleme zu untersuchen

Microbenchmark Code

- 02_java12-17 de.zettsystems.microbenchmark

Benchmark Result - Array

Bench	0	1	10	100
arrayNew	(2,756, 4,460, 14,412), stdev = 1,668	(10,636, 14,387, 24,870), stdev = 2,660	(16,864, 29,366, 682,340), stdev = 66,262	(108,100, 131,660, 247,779), stdev = 24,271
simple	(3,492, 4,192, 12,730), stdev = 1,138	(8,405, 9,216, 22,331), stdev = 1,435	(11,569, 15,732, 193,210), stdev = 18,474	(66,482, 108,830, 798,137), stdev = 79,292
sized	(3,050, 3,552, 5,986), stdev = 0,554	(9,432, 10,823, 19,263), stdev = 1,243	(18,432, 21,171, 50,855), stdev = 4,131	(110,935, 122,572, 168,551), stdev = 11,204
Zero	(2,663, 4,159, 45,382), stdev = 4,441	(10,856, 12,610, 55,991), stdev = 4,586	(17,075, 19,529, 47,148), stdev = 3,451	(110,070, 165,873, 1136,497), stdev = 150,171

Benchmark Result - NumberVerification

Bench	False	True
Regex	0,886 s/op	0,899 s/op
Try/Catch	12,290 s/op	0,227 s/op
StringUtils	0,212 s/op	0,179 s/op

jpackage – Packaging Tool (Java 16)

- Endlich eine einfache Möglichkeit, Java Programme als ausführbare Dateien auszuliefern
- Windows, Linux, Mac werden unterstützt
- Die runtime wird mit ausgeliefert
  - Möchte man eine spezielle, optimierte Runtime (z.B. mit jlink erstellte), muss man die mit –runtime-image angeben
- Für Windows hat man eine ganz normale .exe Datei
  - Entweder als “richtige” .exe bzw. .msi Installer, dann muss man type exe nehmen, aber auch das WiX-tool im path haben (Version 3, jpackage kann mit 4 noch nicht umgehen)
  - Oder als starter-exe mit jar etc. dabei mit type app-image
- Optionen unterscheiden sich zwischen oldschool und modular

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-17
- Neues in der JVM 12-17
- Ausblick 18-21
  - **Pattern Matching for Switch**
  - Record Patterns
  - Virtual Threads
  - Sequenced Collection
- OpenRewrite

Pattern Matching for switch (Java 21)

- Mit dem neuen Feature kann man nun ein Switch mit jedem Typ machen
- Die Typprüfung ist Teil des case
- Da alle möglichen Werte berücksichtigt werden müssen: default muss mit dabei sein (außer bei Sealed Classes Hierarchien)
- Genau wie bei instanceof kann man im Erfolgsfall weitere Prüfungen auf dem Typ vornehmen (Guarded Pattern)

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-17
- Neues in der JVM 12-17
- Ausblick 18-21
  - Pattern Matching for Switch
  - **Record Patterns**
  - Virtual Threads
  - Sequenced Collection
- OpenRewrite

Record Patterns (Java 21)

- „Zerlegung“ von Records beim Pattern Matching
- Wenn man nach dem „cast“ auf die Attribute des Records zugreifen möchte, gibt es jetzt eine Schreibweise, diese direkt als flow Variablen zu deklarieren

Pattern Matching for switch/ Record Pattern Code

- 03_java18-21 de.zettsystems.patternmatching

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-21
- Neues in der JVM 12-17
- Ausblick 18-21
  - Pattern Matching for Switch
  - Record Patterns
  - **Virtual Threads**
  - Sequenced Collection
- OpenRewrite

Virtual Threads (Java 21)

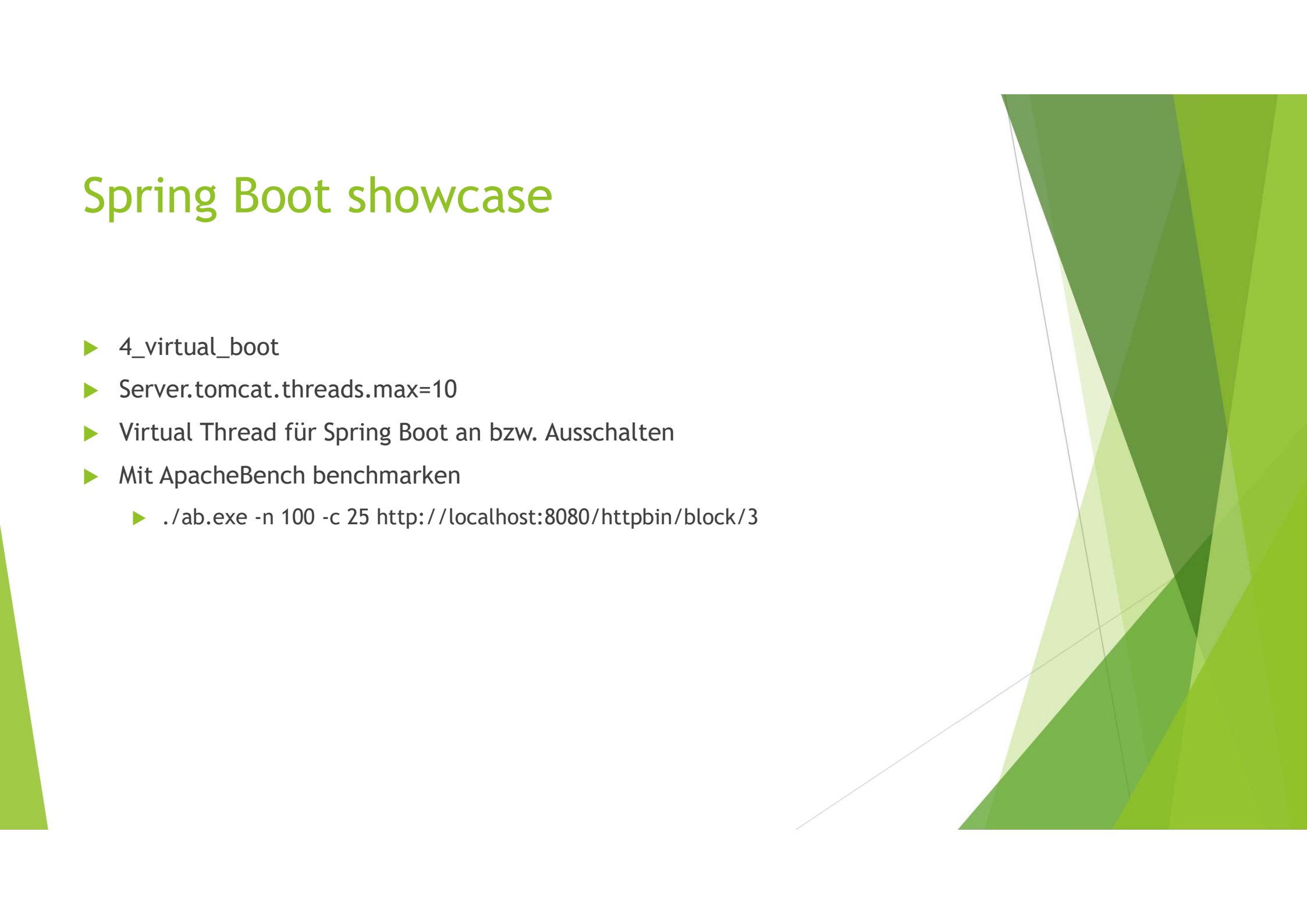
- Unsere bekannten Threads sind dünne Wrapper um einen Betriebssystem Prozess. Diesen nehmen sie sich und behalten ihn für ihre gesamte Lebenszeit, auch wenn sie blockieren und warten (z.B. auf Antwort von einem remote System). Diese threads werden (jetzt) „platform threads“ genannt.
- Die neuen virtual threads nehmen sich den nächstbesten freien OS-Prozess, wenn sie Arbeit erledigen. Wenn sie eine blockierende Aktion aufrufen, werden sie pausiert und der OS-Prozess, den sie gerade nutzten, wird wieder frei.
- → Die Anzahl an platform threads wird durch die Anzahl an OS-Prozessen limitiert. Sie sind eine wertvolle Ressource (Pooling). Virtual threads sind frei davon und daher gut für Applikationen geeignet, die viele Aktionen parallel auszuführen haben.

Virtual Threads Test

- 03_java18-21 test/java/de.zettsystems.virtual

Spring Boot showcase

- 4_virtual_boot
- Server.tomcat.threads.max=10
- Virtual Thread für Spring Boot an bzw. Ausschalten
- Mit ApacheBench benchmarken
  - ./ab.exe -n 100 -c 25 http://localhost:8080/httpbin/block/3



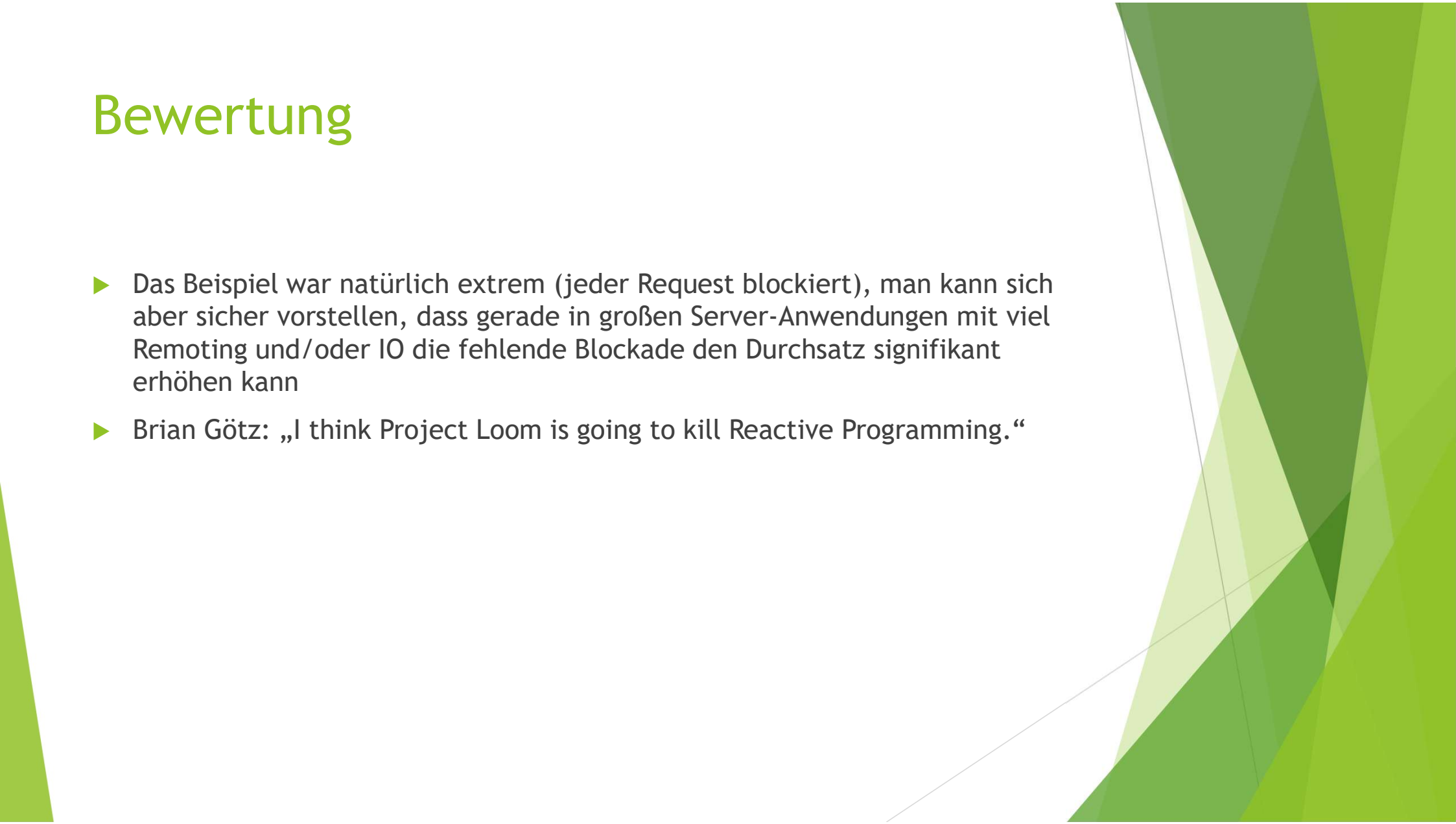
Platform vs. Virtual threads

```
Server Hostname:        localhost
Server Port:            8080

Document Path:          /httpbin/block/3
Document Length:        39 bytes

Concurrency Level:      25
Time taken for tests:   33.124 seconds
Complete requests:      100
Failed requests:        10
   (Connect: 0, Receive: 0, Length: 10, Exceptions: 0)
Total transferred:      17210 bytes
HTML transferred:       3910 bytes
Requests per second:    3.02 [#/sec] (mean)
Time per request:       8281.040 [ms] (mean)
Time per request:       331.242 [ms] (mean, across all concurrent requests)
Transfer rate:          0.51 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.5      0       1
Processing:  3009 6894 1972.3   6027    9041
Waiting:     3006 6892 1972.7   6024    9039
Total:       3010 6894 1972.2   6028    9041

Percentage of the requests served within a certain time (ms)
  50%   6028
  66%   9023
  75%   9026
  80%   9030
  90%   9035
  95%   9037
  98%   9039
  99%   9041
 100%   9041 (longest request)
```

```
Server Hostname:        localhost
Server Port:            8080

Document Path:          /httpbin/block/3
Document Length:        68 bytes

Concurrency Level:      25
Time taken for tests:   15.193 seconds
Complete requests:      100
Failed requests:        91
   (Connect: 0, Receive: 0, Length: 91, Exceptions: 0)
Total transferred:      20281 bytes
HTML transferred:       6981 bytes
Requests per second:    6.58 [#/sec] (mean)
Time per request:       3798.347 [ms] (mean)
Time per request:       151.934 [ms] (mean, across all concurrent requests)
Transfer rate:          1.30 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.5      0       1
Processing:  3006 3017  12.3   3018    3120
Waiting:     3003 3015  11.5   3016    3112
Total:       3006 3018  12.3   3019    3121

Percentage of the requests served within a certain time (ms)
  50%   3019
  66%   3020
  75%   3021
  80%   3022
  90%   3026
  95%   3027
  98%   3029
  99%   3121
 100%   3121 (longest request)
```

Bewertung

- Das Beispiel war natürlich extrem (jeder Request blockiert), man kann sich aber sicher vorstellen, dass gerade in großen Server-Anwendungen mit viel Remoting und/oder IO die fehlende Blockade den Durchsatz signifikant erhöhen kann
- Brian Götz: „I think Project Loom is going to kill Reactive Programming.“



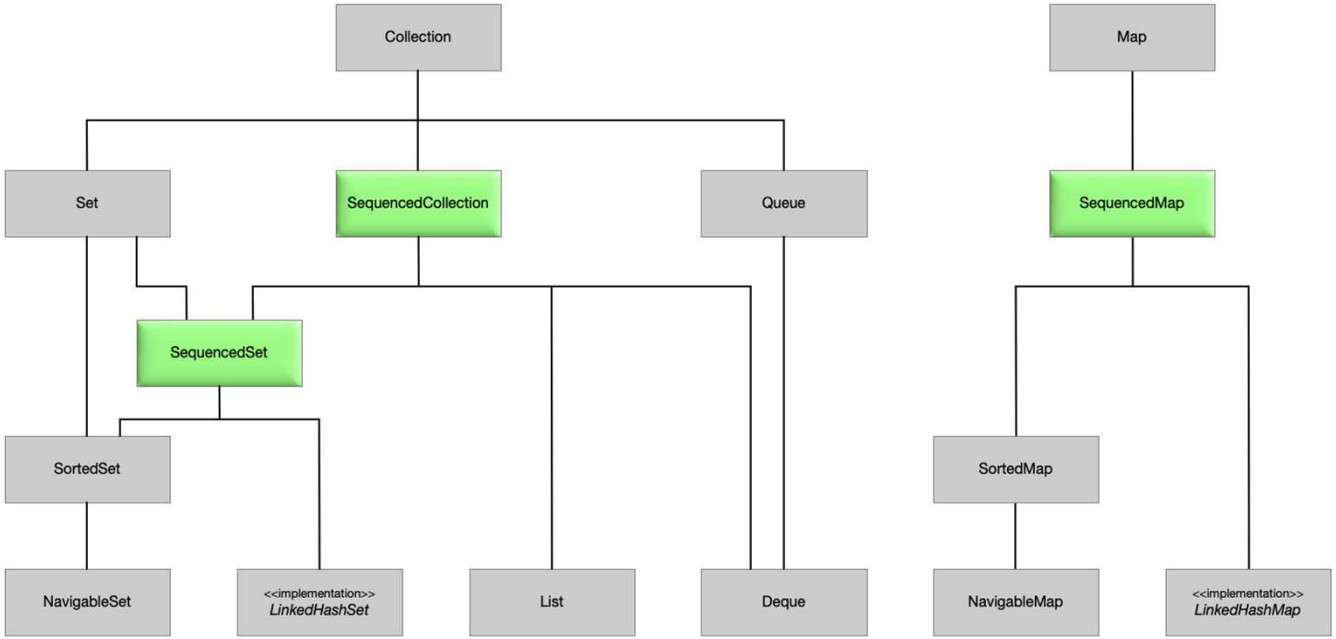
Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-21
- Neues in der JVM 12-17
- Ausblick 18-21
  - Pattern Matching for Switch
  - Record Patterns
  - Virtual Threads
  - **Sequenced Collection**
- OpenRewrite

Sequenced Collections (Java 21)

- Neue interfaces in Collections, die geordnete Elemente darstellen
- Einheitlicher Zugriff auf 1. und letztes Element
- Einheitliches Durchgehen der Elemente vorwärts und rückwärts

Sortiert sich in alte Strukturen ein



Sequenced Collections JEP – Stuart Marks

2022-02-16

Sequenced Collection Code

- 03_java18-21 de.zettsystems.seqcol

Java 18-21 – weitere Verbesserungen

- Parallel und Serial können nun auch -XX:+UseStringDeduplication benutzen
  - Es gibt eine große Menge doppelter String Objekte im Heap
  - Mit obiger Einstellung werden diese beim GC abgeräumt
    - Das gibt bis zu 13% Platz frei!
    - Das kostet aber auch Latenz, also es dauert
- ZGC ist nun auch ein Generational Garbage Collector
  - Generational GC bislang nur bei den anderen GC, reduziert die Zeit, da ältere Objekte nicht jedes Mal angeschaut werden
  - Muss man für Z extra einschalten in Java 21: -XX:+ZGenerational

Verschiedenes

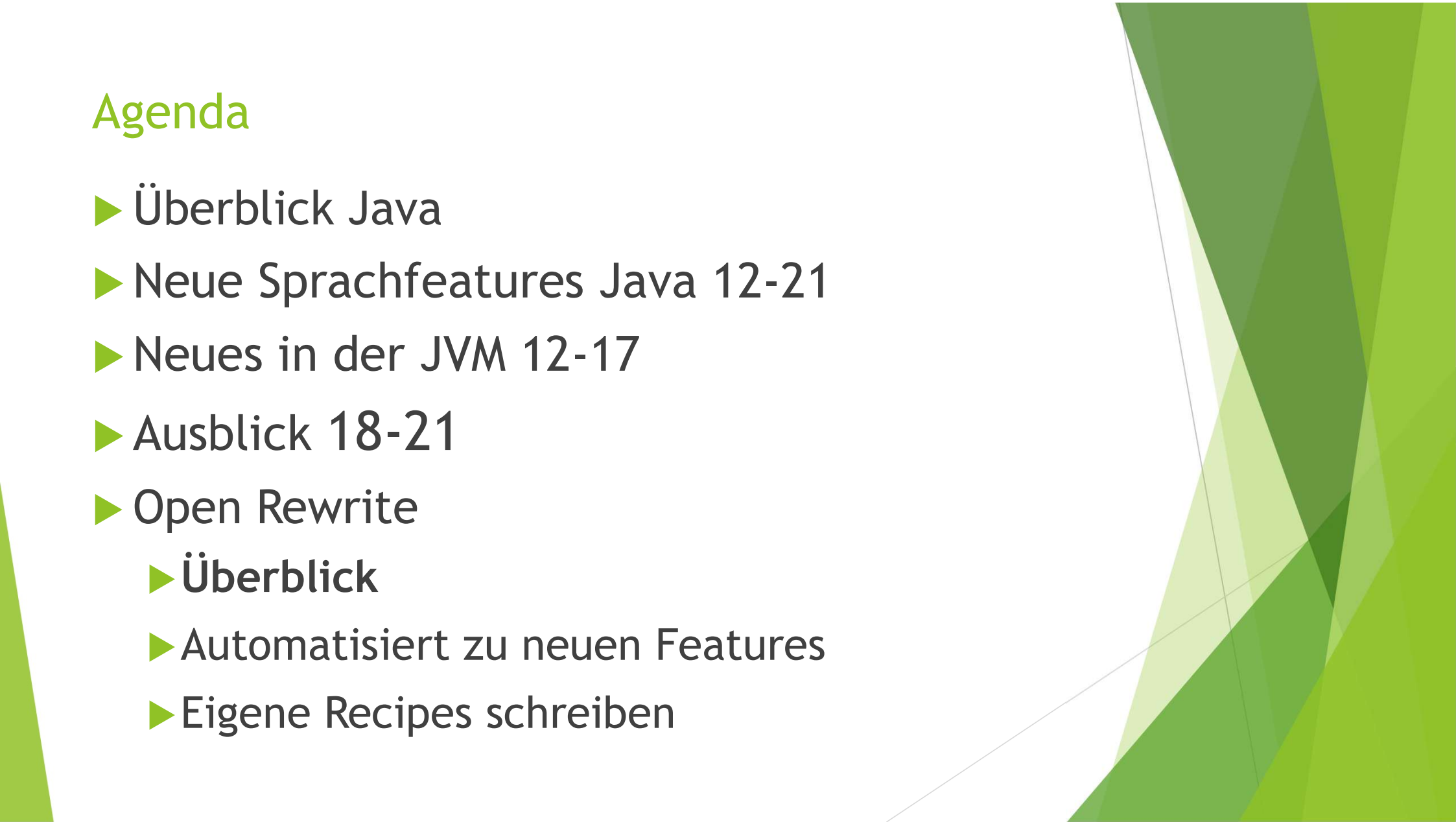
- 03_java18-21 de.zettsystems.history : Können wir das Eingangsbeispiel verbessern?

Ggf. Aufgaben

- Bitte alle TODOs in 03_java_18_21 package de.zettsystems.exercises lösen

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-21
- Neues in der JVM 12-17
- Ausblick 18-21
- Open Rewrite
  - **Überblick**
  - Automatisiert zu neuen Features
  - Eigene Recipes schreiben



Automatisiert Refactorings anwenden

Refactoring: a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. (Fowler, https://martinfowler.com/bliki/DefinitionOfRefactoring.html)

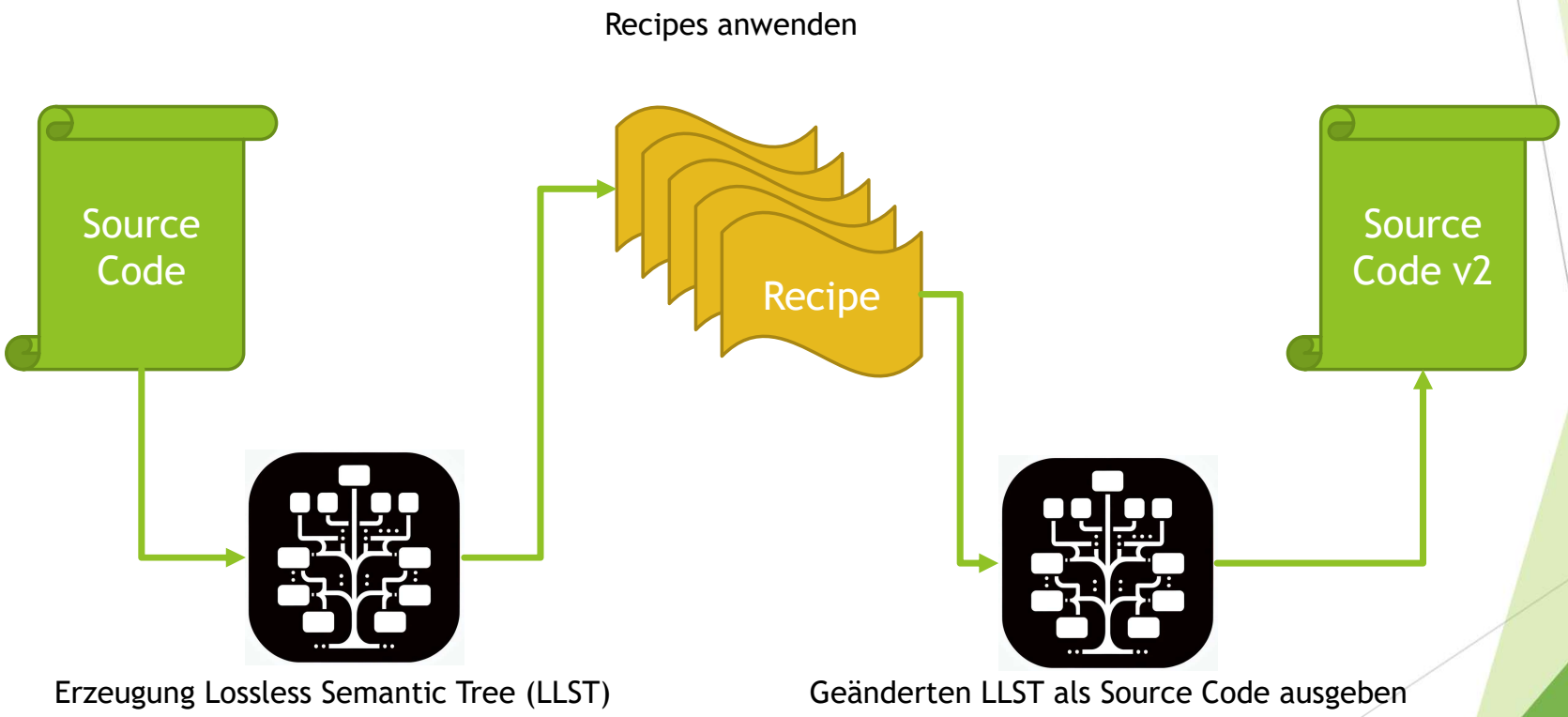
Automatisiert: OpenRewrite liefert Plugins für Maven und gradle und bringt bereits diverse sogenannte Recipes mit, so dass auf die Entwicklerin lediglich die Konfiguration dieser sowie das Review der Ergebnisse zukommt.

Beispiel (oder anders rum ;-)):

```
// Before OpenRewrite
import org.junit.Assert;
...

Assert.assertTrue(condition);
```

```
// After OpenRewrite
import static org.junit.Assert.assertTrue;
...

assertTrue(condition);
```

Funktionsweise



Was gibt es für Recipes („fachlich“)?

Code Style:

- im Prinzip vergleichbar zu PMD, Checkstyle, Sonarqube – nur dass etwaige Verstöße nicht angemakert, sondern gleich behoben werden. Z.B.:
  - org.openrewrite.staticanalysis.RemoveUnusedPrivateFields
  - org.openrewrite.staticanalysis.UnnecessaryParentheses

Updates:

- Durchführung von Java und Library Updates!
  - org.openrewrite.java.migrate.UpgradeToJava17
  - org.openrewrite.java.testing.junit5.JUnit5BestPractices
  - org.openrewrite.java.testing.assertj.Assertj

Typen von Recipes

- **Deklarative Recipes**: Zusammenstellung vorhandener Recipes und ggf. deren Parametrierung in der rewrite.yml
- **Refaster Templates:** Einfache Eine-Zeile-Ersetzungen
- **Imperative Recipes:** Ein Programmiertes Recipe

```yaml
---
type: specs.openrewrite.org/v1beta/recipe
name: org.openrewrite.staticanalysis.CommonStaticAnalysis
displayName: Common static analysis issues
description: Resolve common static analysis issues discovered through 3rd party tools.
recipeList:
  - org.openrewrite.staticanalysis.AtomicPrimitiveEqualsUsesGet
  - org.openrewrite.staticanalysis.BigDecimalRoundingConstantsToEnums
  - org.openrewrite.staticanalysis.BooleanChecksNotInverted
  - org.openrewrite.staticanalysis.CaseInsensitiveComparisonsDoNotChangeCase
  - org.openrewrite.staticanalysis.CatchClauseOnlyRethrows
  - org.openrewrite.staticanalysis.ChainStringBuilderAppendCalls
```

```java
public static class RedundantCall {
    @BeforeTemplate
    public String start(String string) {
        return string.substring(0, string.length());
    }

    @BeforeTemplate
    public String startAndEnd(String string) {
        return string.substring(0);
    }

    @BeforeTemplate
    public String toString(String string) {
        return string.toString();
    }

    @AfterTemplate
    public String self(String string) {
        return string;
    }
}
```

Agenda

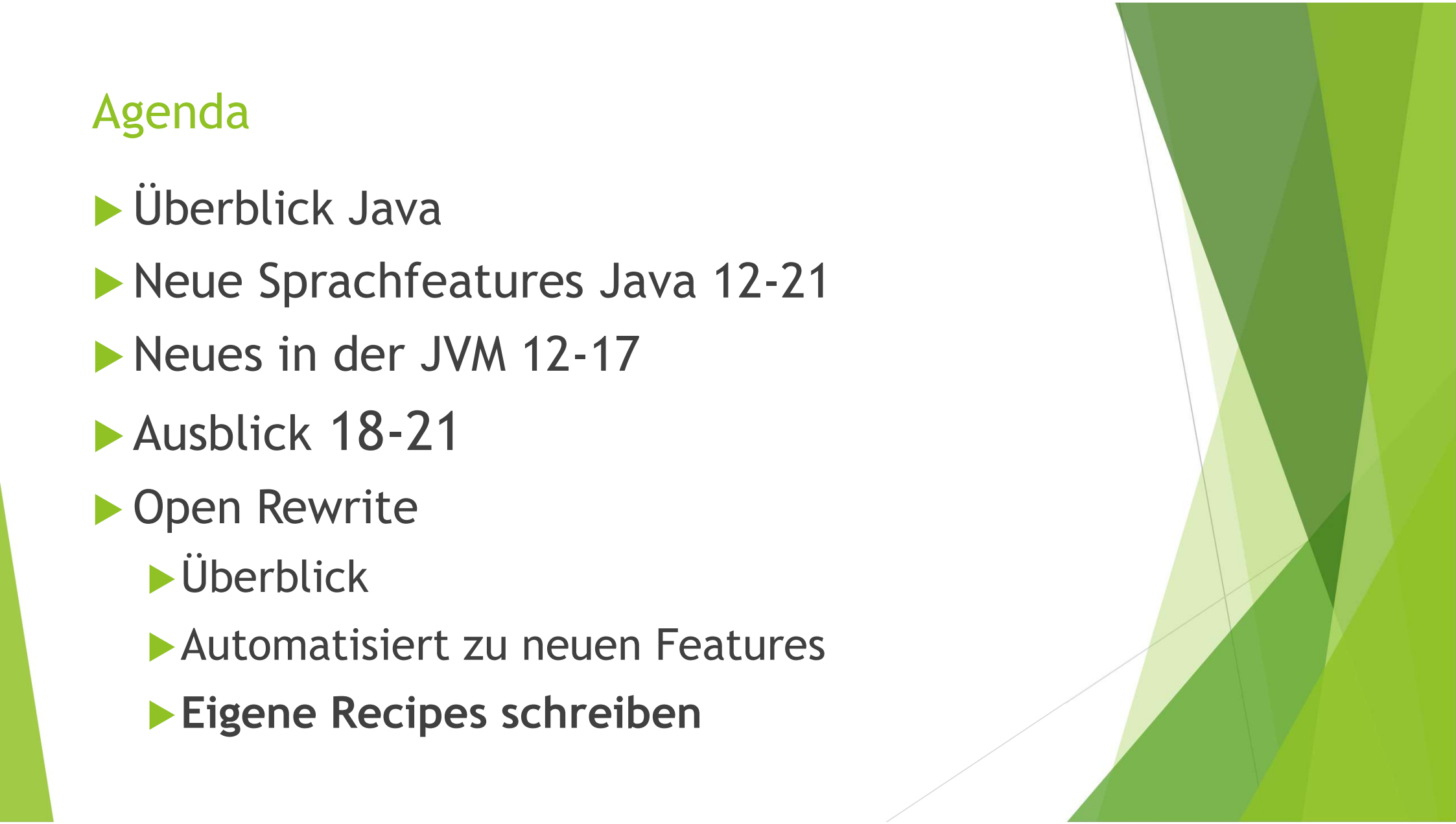
- Überblick Java
- Neue Sprachfeatures Java 12-21
- Neues in der JVM 12-17
- Ausblick 18-21
- Open Rewrite
  - Überblick
  - **Automatisiert zu neuen Features**
  - Eigene Recipes schreiben

Altanwendung automatisiert modernisieren

- 05_openrewrite
- https://docs.openrewrite.org/recipes/java/migrate/javaversion17
  - TextBlocks
  - PatternMatching InstanceOf
  - Stream.toList() Recipe muss man zusätzlich nennen

Agenda

- Überblick Java
- Neue Sprachfeatures Java 12-21
- Neues in der JVM 12-17
- Ausblick 18-21
- Open Rewrite
  - Überblick
  - Automatisiert zu neuen Features
  - **Eigene Recipes schreiben**



Zettsystems-recipes

- https://github.com/MichaelZett/zettsystems-recipes

Abschluss

- Fragen?