Michael Inden

Workshop: Best of Java 8 und 9

IT-Tage 2016 | 12.-15.Dezember 2016 | Frankfurt am Main | Maritim Hotel

JDK 9 IM ÜBERBLICK

INHALT

- Schedule und IDE-Support
- Syntax-Erweiterungen in Java 9
- Neuheiten und Änderungen im JDK
- JShell - REPL mit Java
- Modularisierung mit Project Jigsaw
- Sonstiges

SCHEDULE UND IDE-SUPPORT

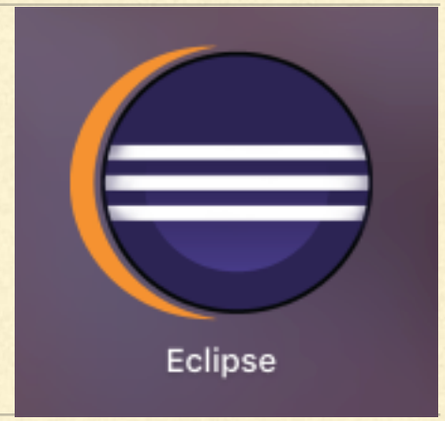
JDK 9 SCHEDULE

- **2016/05/26 Feature Complete**
- 2016/08/11 All Tests Run
- 2016/10/20 Zero Bug Bounce
- 2017/01/26 Final Release Candidate
- **2017/03/23 General Availability (Verschoben)**
- **2017/07/27 General Availability (Wirklich ??)**

JDK 9 SCHEDULE

- Rund 3,5 Jahre nach JDK 8 kommt nun Java 9 im Juli 2017 ?!
- JDK 8: Fokus funktionale Programmierung, Lambdas & Streams
- **JDK 9: Steht im Zeichen der Modularisierung**
- Module als eigenständige Softwarekomponenten
- Bis dato viele Probleme von Oracle zu lösen

IDE-SUPPORT



- **IntelliJ 2016.2** läuft problemlos mit JDK 9, weil es eine eigene Java 8 Runtime mitbringt

- **Eclipse Oxygen** mit zusätzlichem JDK 9 Patch
  http://marketplace.eclipse.org/content/java-9-support-beta-neon

  Achtung: Startet nur, wenn Eclipse.ini
  a) auf die JDK 8 VM zeigt:
  -vm
  /Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/Home/bin
  b) mit --add-modules=java.se.ee

SYNTAX-ERWEITERUNGEN

Anonyme innere Klasse und der Diamond Operator

- JDK 8:

```
final Comparator<String> byLength = new Comparator<String>()
{
    ...
};
```

- JDK 9

```
final Comparator<String> byLength = new Comparator<>()
{
    ...
};
```

Effectively final Variablen im ARM

- JDK 8:

```
final BufferedInputStream bufferedIs = ...;
BufferedOutputStream bufferedOs = ...

try (final BufferedInputStream bis = bufferedIs;
     final BufferedOutputStream bos = bufferedOs)
{
    ...
```

- JDK 9

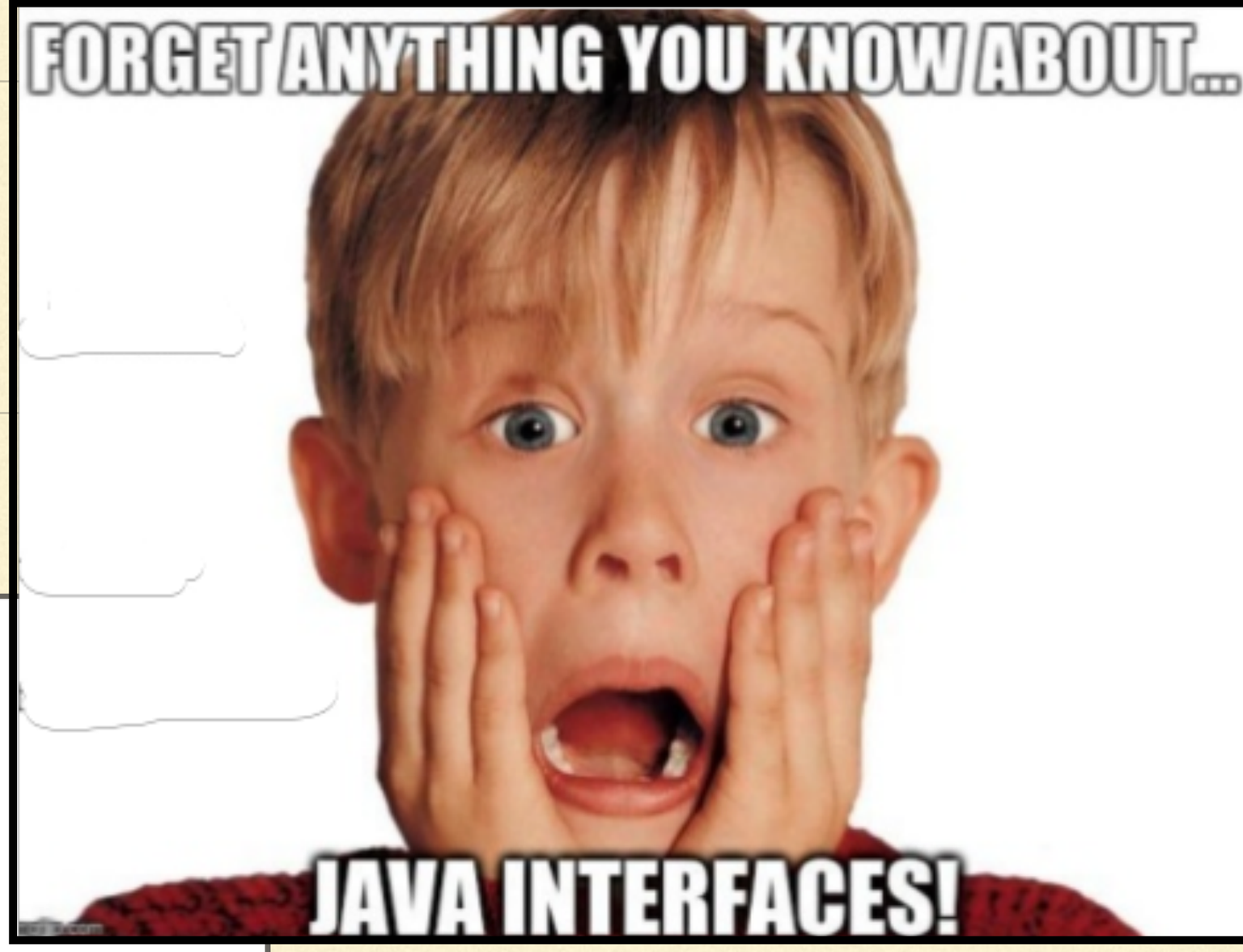
```
final BufferedInputStream bufferedIs = ...;
BufferedOutputStream bufferedOs = ...;

try (bufferedIs;
      bufferedOs)
{
    ...
```

Erweiterungen in der @Deprecated-Annotation

- @Deprecated dient zum Markieren von obsoletem Sourcecode
- JDK 8: ohne Parameter
- JDK 9: Zwei Parameter @since und @forRemoval

```
/**
* @deprecated this method is replaced by someNewMethod() which is
more stable
*/
@Deprecated(since = "7.2", forRemoval = true)
private static void someOldMethod()
{
   // ...
}
```




```
public interface PrivateMethodsExample
{
    public abstract int method1();

    public default int calc(int a, int b) {
        return myCalc(a, b);
    }

    public default int calc2(int a, int b) {
        return myCalc(a, b);
    }

    private int myCalc(int a, int b) {
        return a + b;
    }
}
```

Underscore als Identifier

- _ ist kein valider Bezeichner mehr
(semantisch war er es aber eh nie ;-))

- final String _ = "Underline";

```
MyClass.java:2: error: as of release 9, '_' is a keyword, and may not be
used as an identifier
    static Object _ = new Object();
                  ^
```

NEUHEITEN UND ÄNDERUNGEN IM JDK

- Process API
- Stream-API
- Optional<T>
- Collection-Factory-Methods
- HTTP 2
- Reactive Streams

PROCESS API

PROCESS API

- Bisher nur begrenzte Kontrolle und Verwaltung von Betriebssystemprozessen => Beispiel PID auslesen
- Oftmals für jede Plattform eine andere Implementierung

```
private static long getPidOldSytle() throws InterruptedException, IOException
{
    final Process proc = Runtime.getRuntime().exec(new String[]{ "/bin/sh", "-c", "echo $PPID" });
    if (proc.waitFor() == 0)
    {
        final InputStream in = proc.getInputStream();
        final byte[] outputBytes = new byte[in.available()];

        in.read(outputBytes);
        final String pid = new String(outputBytes);
        return Long.parseLong(pid.trim());
    }
    throw new IllegalStateException("PID is not accessible");
}
```

PROCESS API

- Neu: Betriebssystemunabhängige Methoden für PIDs, Prozessnamen und Zustände
- System.out.println(“PID: ” + **ProcessHandle.current().getPid()**);

```
final ProcessHandle current = ProcessHandle.current();
printInfo(current);

private static void printInfo(final ProcessHandle current)
{
    System.out.println("PID: " + current.getPid());
    System.out.println("Info: " + current.info());
    System.out.println("Command: " + current.info().command());
    System.out.println("CPU-Usage: " + current.info().totalCpuDuration());
}
```

STREAM API

STREAM-API: TAKEWHILE() / DROPWHILE()

- Das umfangreiche Stream-API war eine der wesentlichen Neuerungen in Java 8
- takeWhile(Predicate<T>) – Verarbeitet Elemente des Streams, solange die übergebene Bedingung erfüllt ist.
- dropWhile(Predicate<T>) – Überspringt Elemente des Streams, solange die übergebene Bedingung erfüllt ist

```
final IntStream stream1 = IntStream.iterate(1, n -> n + 1);
System.out.println(stream1.takeWhile(n -> n < 10).
                                    mapToObj(Integer::toString).
                                    collect(joining(", "))); // 1, 2, 3, 4, 5, .., 9

final IntStream stream2 = IntStream.rangeClosed(7, 14);
System.out.println(stream2.dropWhile(n -> n < 10).
                                    mapToObj(Integer::toString).
                                    collect(joining(", "))); // 10, 11, 12, 13, 14
```

STREAM-API BEISPIEL

- Kombination der beiden Methoden zur Extraktion von Daten

```
public static void main(final String[] args)
{
    Stream<String> words = Stream.of("ab", "bla", "<START>",
                                     "Hier", "steht", "der", "Text",
                                     "<END>", "saas", "bla");

    Stream<String> content = words.dropWhile(word -> !word.contains("<START>"))
                                  .skip(1)
                                  .takeWhile(word -> !word.contains("<END>"));

    content.forEach(System.out::println);
}
```

```
Hier
steht
der
Text
```

OPTIONAL<T>

OPTIONAL<T>

- Die Klasse Optional<T> wurde mit Java 8 eingeführt und erleichtert die Behandlung und **Modellierung optionaler Werte**.

- Recht gutes API, aber **3 Schwachstellen** bei folgende Aufgabenstellungen:

1. Das Ausführen von Aktionen auch im Negativfall.

2. Die Umwandlung in einen Stream<T> , für eine Kompatibilität mit dem Stream-API z. B. für Frameworks, die auf Streams arbeiten,

3. Die Verknüpfung der Resultate mehrerer Berechnungen, die Optional<T> liefern.

OPTIONAL<T> NEGATIVAKTION

- isPresent() / get() und Negativfall:

```
final Optional<String> optCustomer = findCustomer("UNKNOWN");
if (optCustomer.isPresent())
{
    System.out.println("found: " + optCustomer.get());
}
else
{
    System.out.println("not found“);
}
```

- JDK 9: ifPresentOrElse(Consumer<? super T>, Runnable)

```
final Optional<String> optCustomer2 = findCustomer("UNKNOWN");
optCustomer2.ifPresentOrElse(str -> System.out.println("found: " + str),
                             () -> System.out.println("not found"));
```

OPTIONAL<T> VERKNÜPFUNGEN

- JDK 9: or(Supplier<? extends Optional<? extends T>> supplier)

```
public static void main(final String[] args)
{
    final Optional<String> optCustomer = multiFindCustomer("Tim");
    optCustomer.ifPresentOrElse(str -> System.out.println("found: " + str),
                                () -> System.out.println("not found"));
}

private static Optional<String> multiFindCustomer(final String customerId)
{
    return findInCache(customerId)
                .or(() -> findCustomer(customerId))
                .or(() -> findInDb(customerId));
}
```

COLLECTION FACTORY METHODS

COLLECTION LITERALS **LIGHT**

List<String> names = [„MAX“, „Moritz“, „Tim“];
Map<String, Integer> nameToAge = { „Tim“ : 45, „Tom“ : 23 };

```
List.of(1, 2, 3);

Set.of(“a”, “b”);

Map.of(5, “test”, 6, “test2”)

Map.fromEntries(

    entry(5, “test”),

    entry(6, “test2”));
```

FACTORY METHODS BEHAVIOR

```
List<String> names = List.of("MAX", "MORITZ", "MIKE");
names.forEach(name -> System.out.println(name));

MAX
MORITZ
MIKE
```

```
Set<String> names2 = Set.of("MAX", "MORITZ", "MAX");
names2.forEach(name -> System.out.println(name));


Exception in thread "main" java.lang.IllegalArgumentException:
duplicate element: MAX
    at java.util.ImmutableCollections$SetN.<init>(java.base@9-ea/
ImmutableCollections.java:329)
    at java.util.Set.of(java.base@9-ea/Set.java:500)
    at jdk9example.Jdk9Example.main(Jdk9Example.java:31)
```

FACTORY METHODS IM DETAIL

```
static <E> List<E> of(E... elements) {
        switch (elements.length) {
            case 0:
                return new ImmutableCollections.List0<>();
            case 1:
                return new ImmutableCollections.List1<>(elements[0]);
            case 2:
                return new ImmutableCollections.List2<>(elements[0], elements[1]);
            default:
                return new ImmutableCollections.ListN<>(elements);
        }
}
```

FACTORY METHODS IM DETAIL

- https://www.youtube.com/watch?v=OJrIMv4dAek

#Jav

Space Efficiency

- Consider an unmodifiable set containing two strings

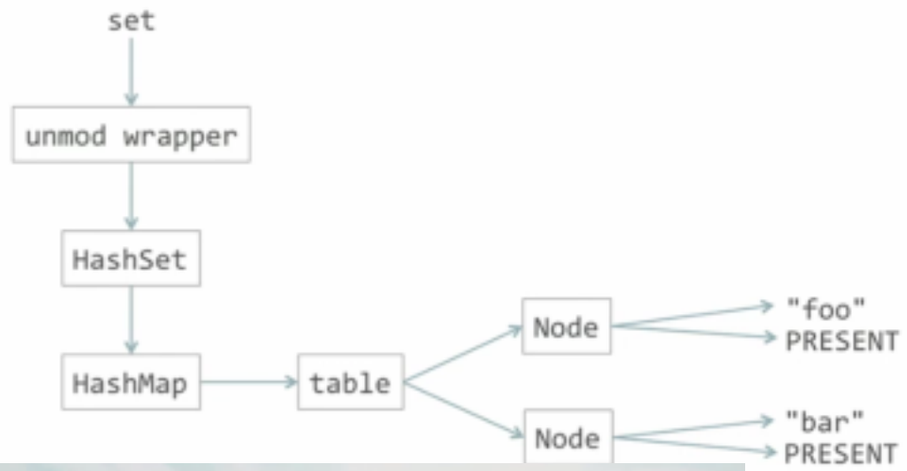
```
Set<String> set = new HashSet<>(3);  // 3 is the number of buckets
set.add("foo");
set.add("bar");
set = Collections.unmodifiableSet(set);
```

- How much space does this take? Count objects.
  - 1 unmodifiable wrapper
  - 1 HashSet
  - 1 HashMap
  - 1 Object[] table of length 3
  - 2 Node objects, one for each element

Java ORACLE

#JavaYoungPup

Space Efficiency



#JavaYoungPups

Space Efficiency

- Proposed field-based set implementation

```
Set<String> set = Set.of("foo", "bar");
```

- One object, two fields
  - 20 bytes, compared to 152 bytes for conventional structure
- Efficiency gains
  - lower fixed cost: fewer objects created for a collection of any size
  - lower variable cost: fewer bytes overhead per collection element

HTTP/2

HTTP/2 — SYNCHRON

- basiert auf Googles SPDY
- beträchtliche Geschwindigkeitsverbesserungen von 10 bis 50 %

```
final URI uri = new URI(„http://www.oracle.com/index.html");
final HttpRequest request = HttpRequest.create(uri).GET();
final HttpResponse response = request.response();

final int responseCode = response.statusCode();
final String responseBody = response.body(asString());

System.out.println("Status: " + responseCode);
System.out.println("Body:   " + responseBody);
```

Charset.forName("UTF-8")

HTTP/2 — ASYNCHRON

```
final URI uri = new URI("http://www.oracle.com/index.html");
final HttpRequest request = HttpRequest.create(uri).GET();
final CompletableFuture<HttpResponse> asyncResponse =

request.responseAsync();

TimeUnit.MILLISECONDS.sleep(500); // or do some work in parallel
if (!asyncResponse.isDone())
{
    asyncResponse.cancel(true);
    System.err.println("timeout");
}
else
{
    printResponseInfo(asyncResponse.get());
}
```

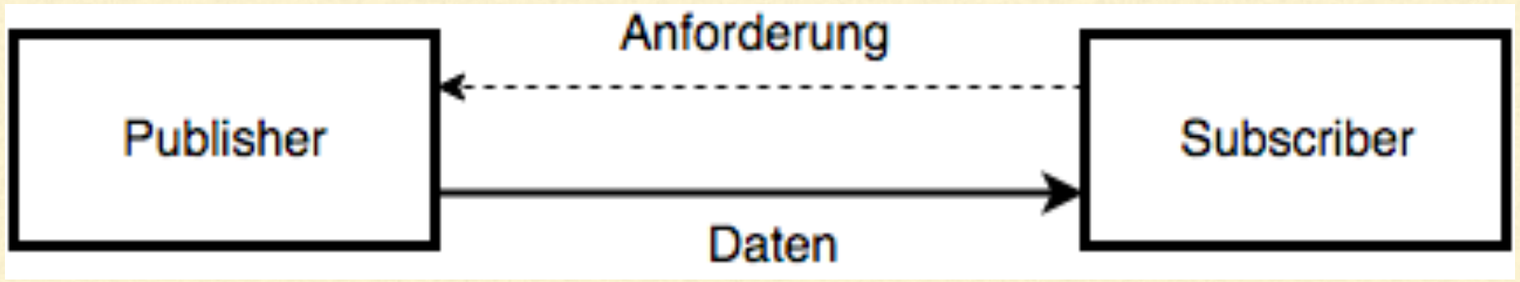
REACTIVE STREAMS

REACTIVE STREAMS

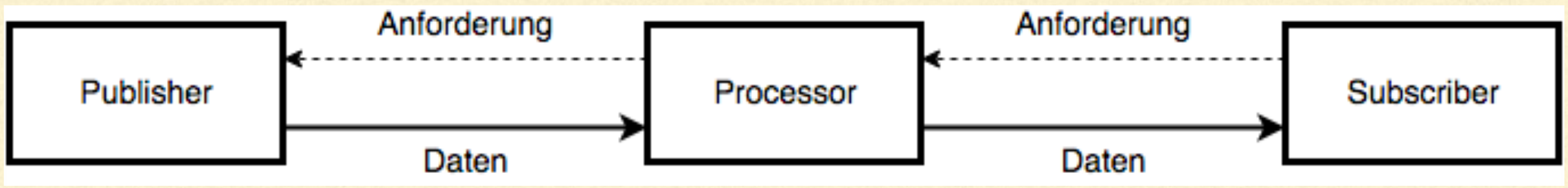
- Asynchrone Stream-Verarbeitung mit Backpressure
- unabhängig ausgeführte Verarbeitungseinheiten kommunizieren über Events bzw. Messages
- JDK 9: **Publish-Subscribe-Framework** durch die Klasse **Flow** sowie eine Utility-Klasse **SubmissionPublisher**
- Frameworks wie Akka, RxJava, Vert.x und Reactor

REACTIVE STREAMS

- Publisher veröffentlichen Daten und Subscriber melden sich an
- Schematische Zusammenarbeit von Publisher und Subscriber



- Ausbau zu Verarbeitungskette



- ähnlich wie bei einem Umzug eine Kette von Helfern

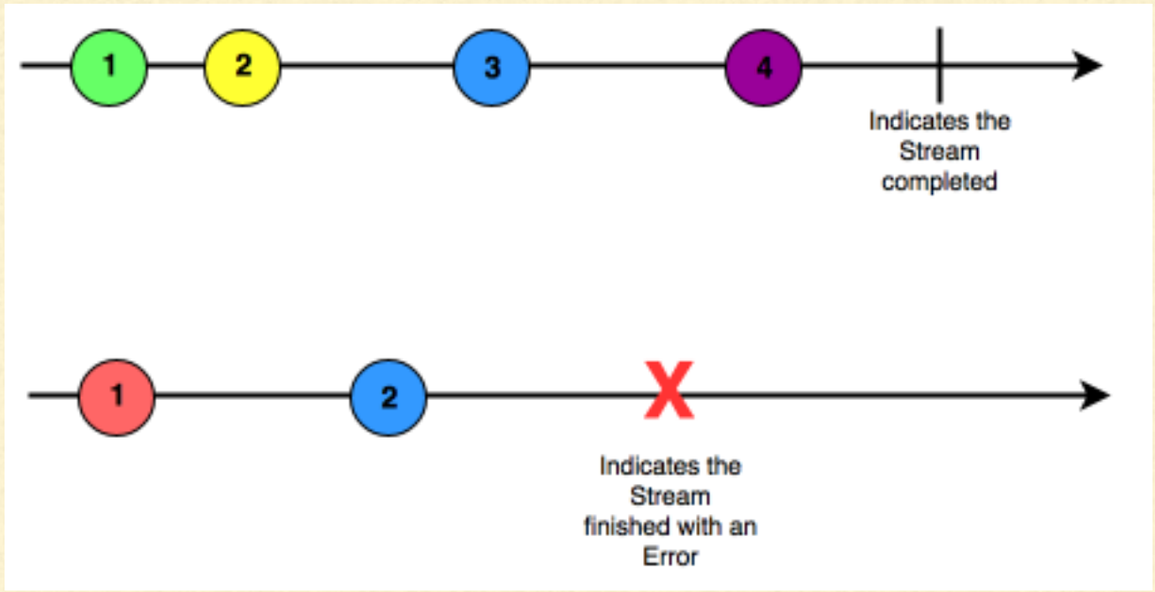
REACTIVE STREAMS

- Engpässe bei Kommunikation, wenn ein **Publisher zu langsam oder zu schnell arbeitet**. => Subscriber teilweise ausgelastet oder überlastet
  - **Eingangspuffer** als Abhilfe, aber größenbeschränkt
  - **Drosselung** des Senders => verschwendet potentiell Performance
- Reactive Streams adressieren die Engpässe im Datenfluss.
  Dazu: Konzept der Backpressure (Gegendruck oder Rückstau)

REACTIVE STREAMS — MARBLE DIAGRAMME

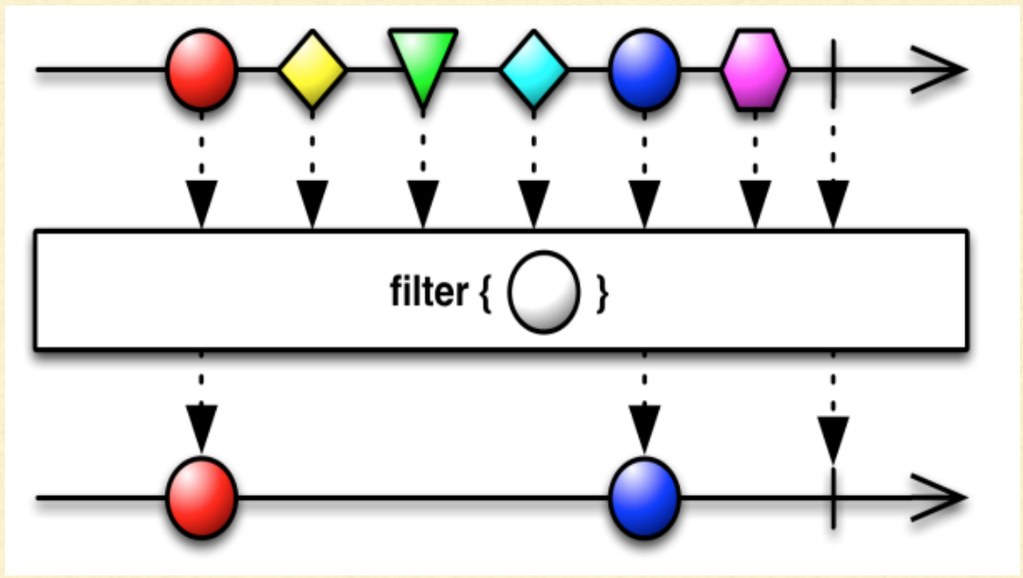
- Stream kann als Producer von 0..N Events und einer von zwei Terminal-Operations (Completed (keine weiteren Daten), Error (Fehler aufgetreten)) angesehen werden

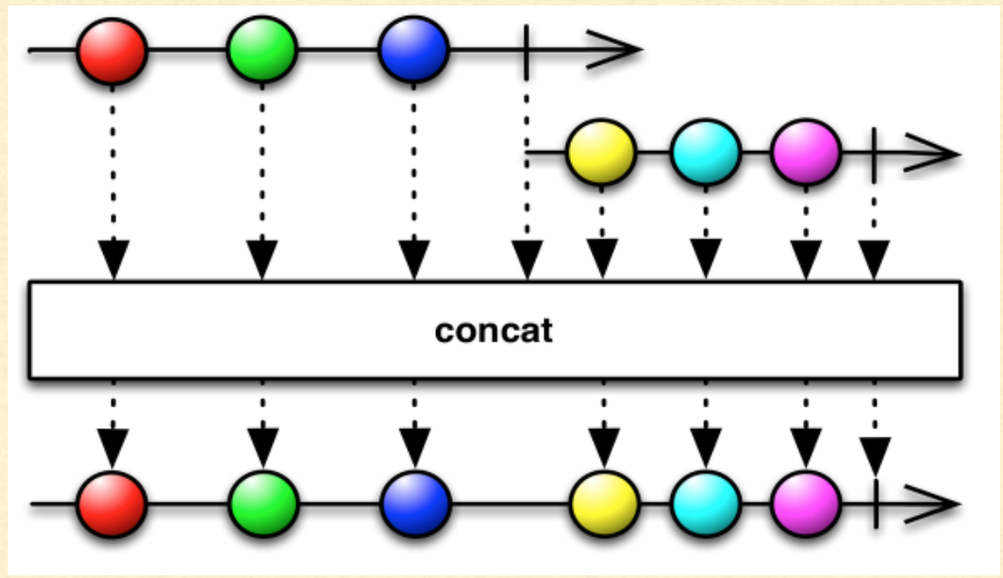
- Visualisierung mit sogenannten 'marble diagrams'

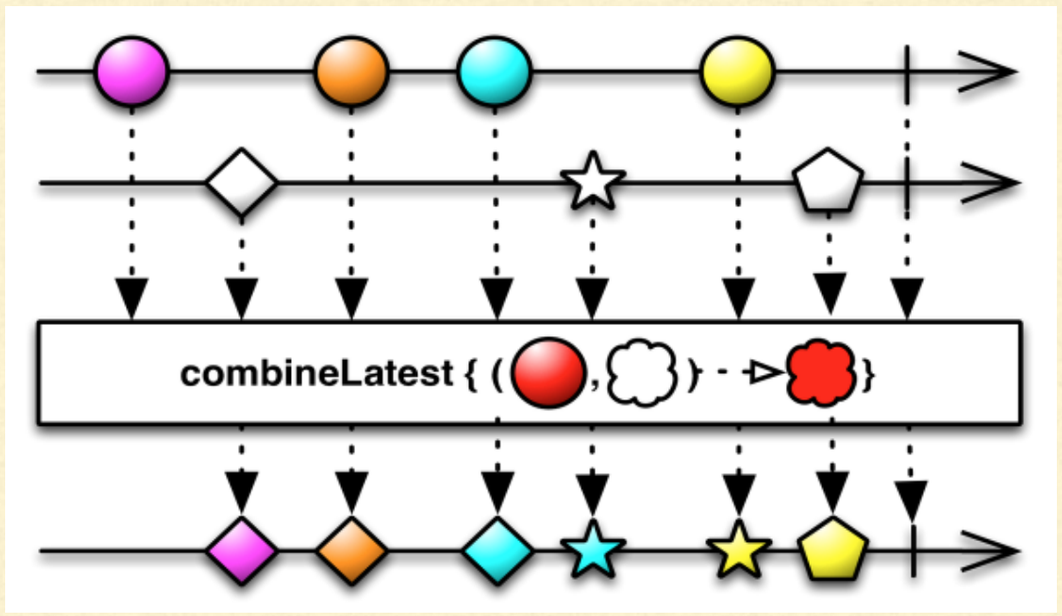


Grafiken von https://balamaci.ro/reactive-log-processing/

REACTIVE STREAMS — MARBLE DIAGRAMME



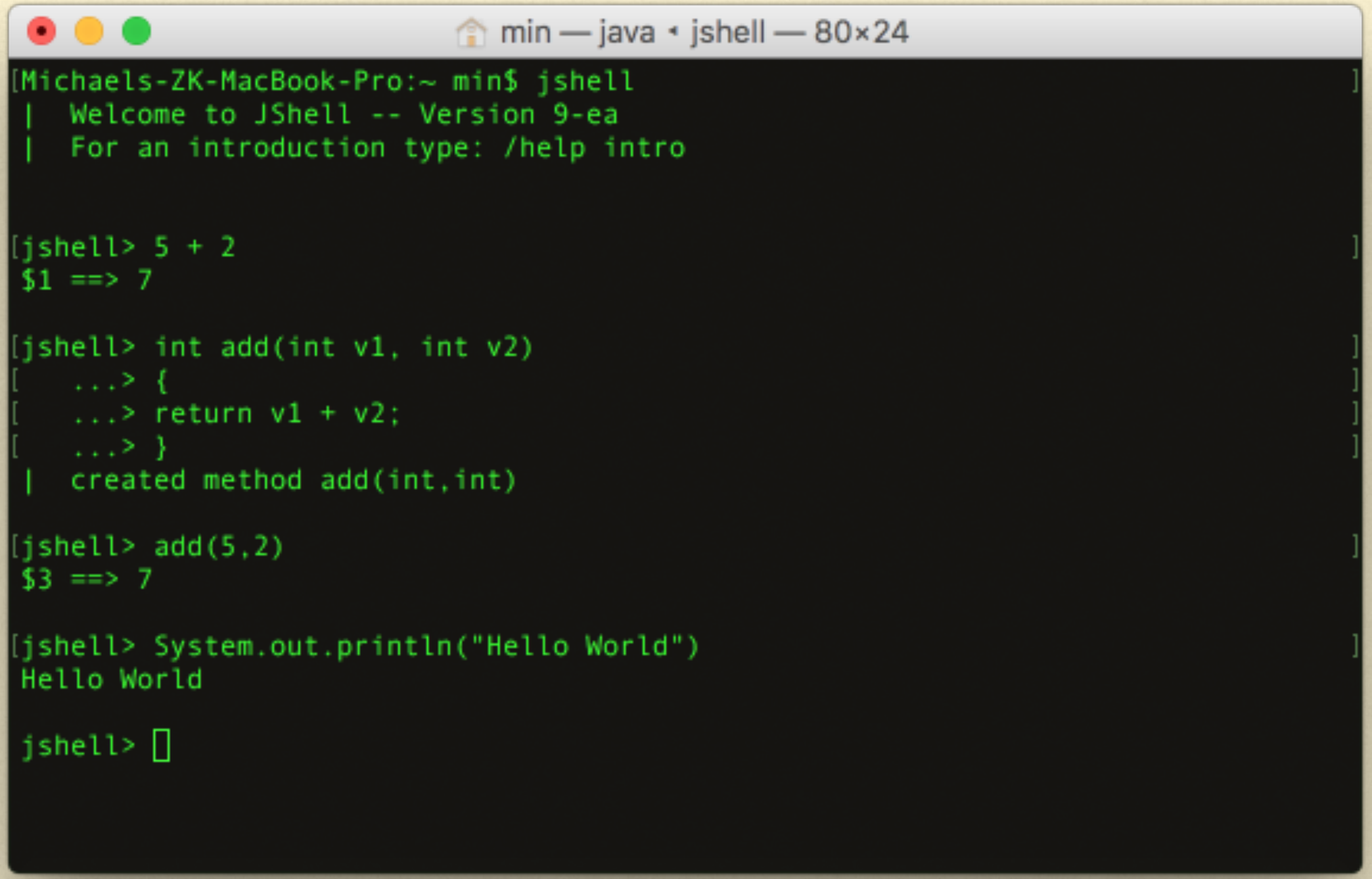




Grafiken von https://balamaci.ro/reactive-log-processing/

JSHELL — REPL MIT JAVA

JAVA + REPL = JSHELL



ÜBUNGEN - Teil I

MODULARISIERUNG

- Einführung in Project JIGSAW
- Beispiel mit 2 Modulen
- Besonderheiten

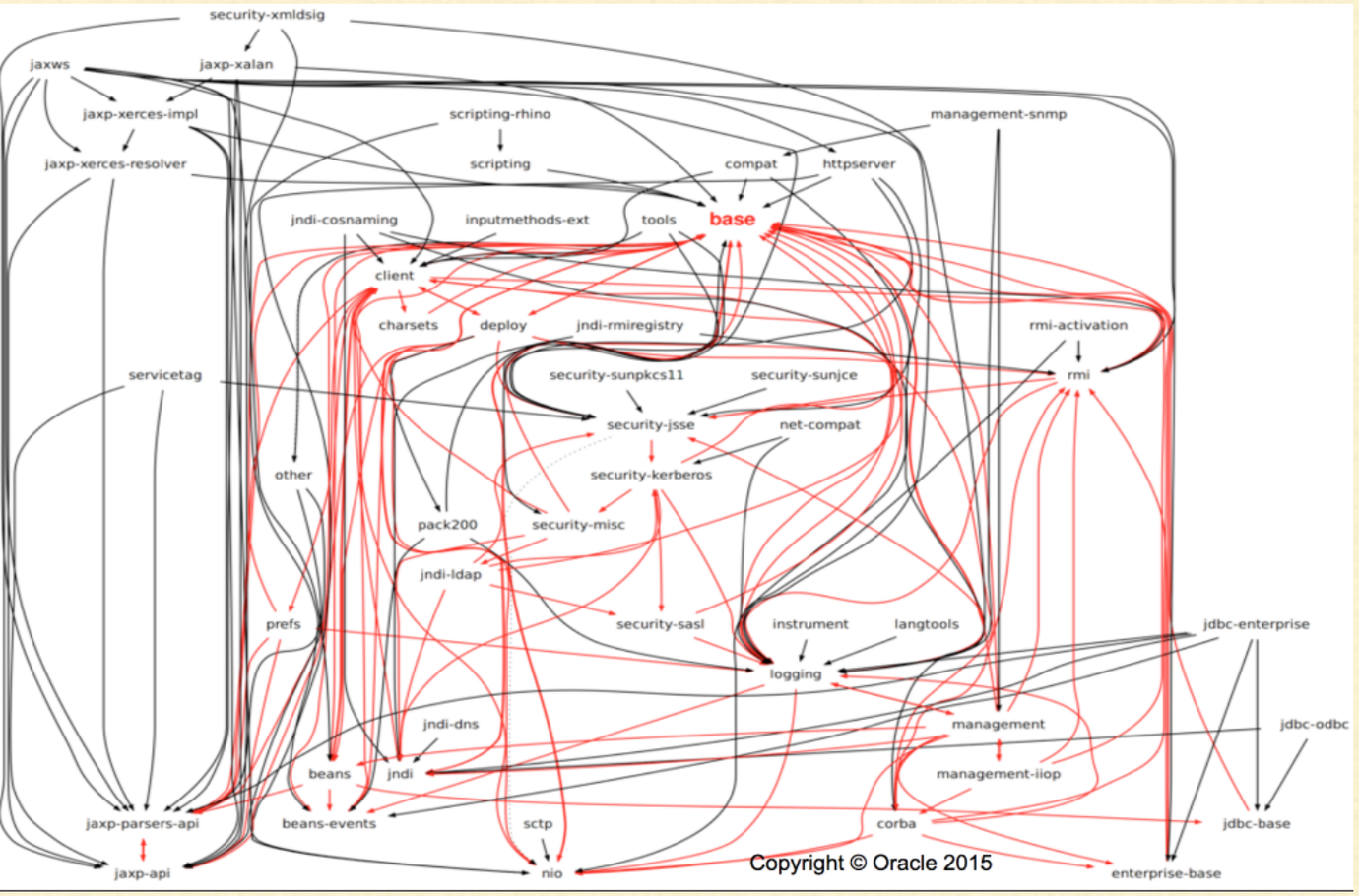
Einführung in Project Jigsaw:

- Ziele
- Warum Modularisierung?
- Modularisierung des JDKs
- Was ist ein Modul?
- Abhängigkeiten und Sichtbarkeiten steuern
- Verzeichnislayout?

ZIELE VON PROJECT JIGSAW

- Modularisierung des JDKs an sich
- Modularisierung von Anwendungen und Bibliotheken ermöglich
- zuverlässige Konfiguration statt fehlerträchtiger Abhängigkeitsverwaltung
- Aber: Warum???

CHAOS IM JDK & CLASSPATH

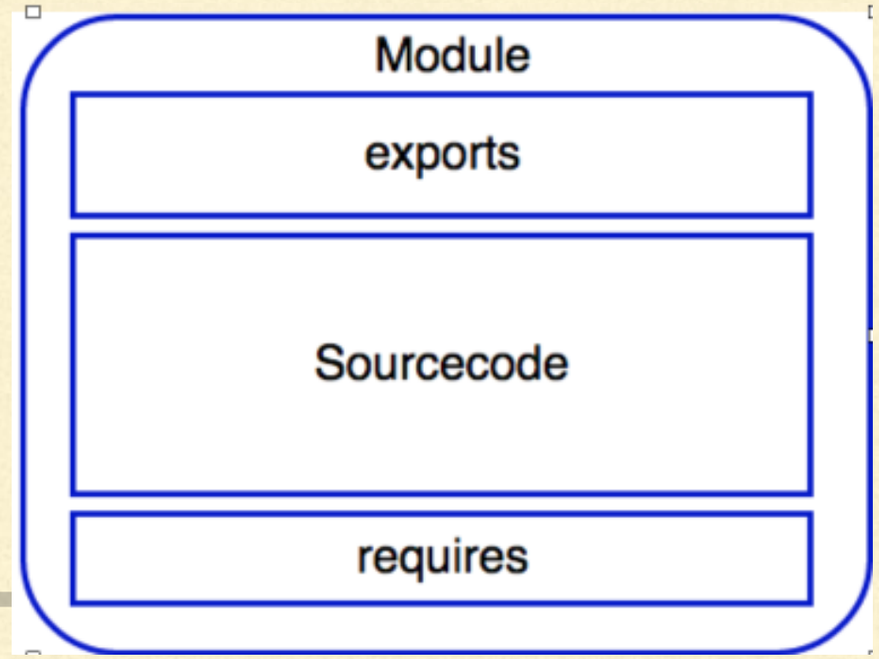


BISHERIGE VARIANTEN

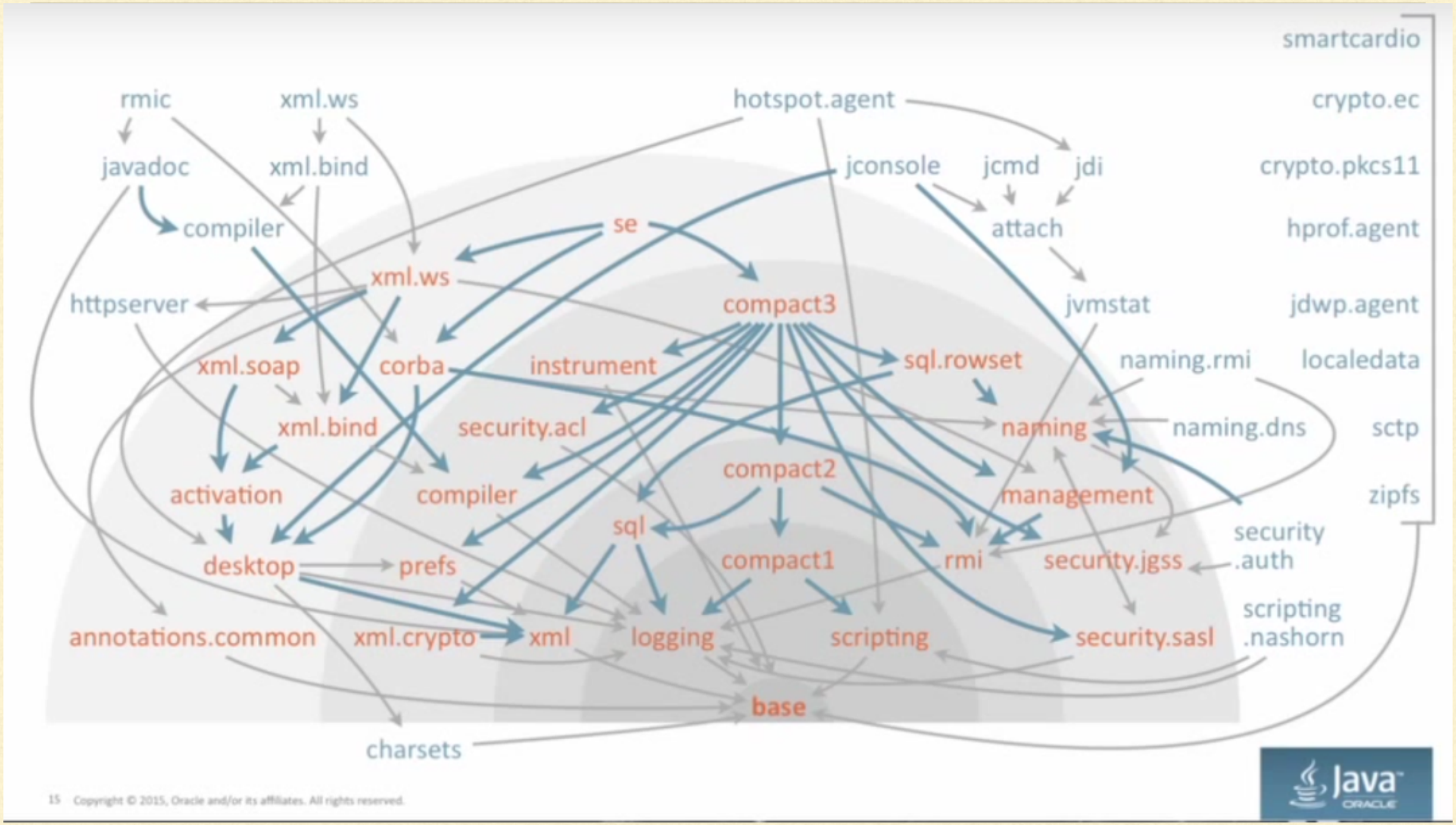
- „Module“ als Sammlung von Klassen in Packages bzw. als JARs
  - lose Kopplung lässt sich nicht forcieren
  - Abhängigkeiten und Sichtbarkeiten schwierig zu steuern
- „Module“ mithilfe von OSGi
  - sehr ausgereift
  - aber etwas komplex und nicht für das JDK geeignet

LÖSUNG: MODULE

- Module als neue Softwarebausteine im JDK
- Ein Modul …
  - besitzt einen eindeutigen Namen
  - abgegrenzte Funktionalität über wohldefinierte Schnittstelle
  - versteckt Implementierungsdetails
  - besitzt klar definierte Abhängigkeiten
  - besteht aus Packages, Klassen usw.



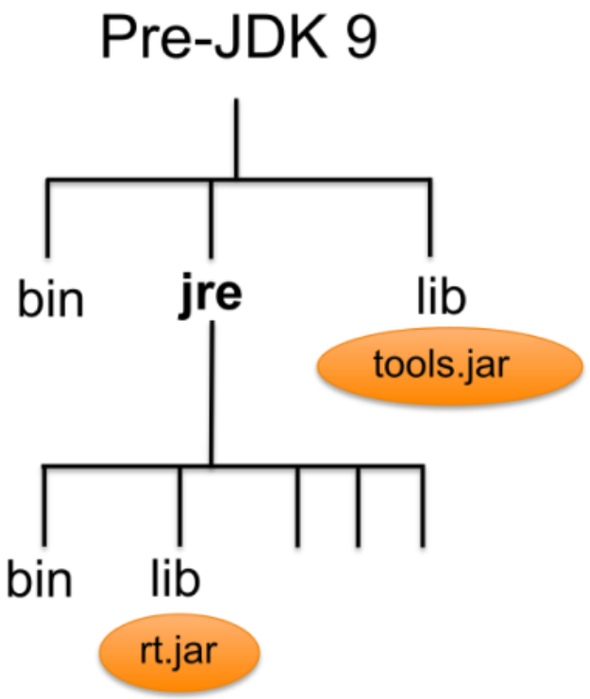
MODULARISIERUNG DES JDKS



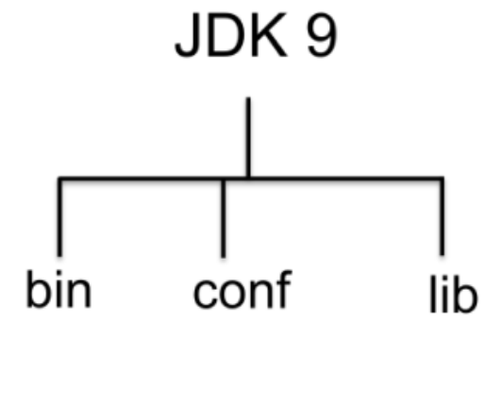
MODULARISIERUNG DES JDKS

- JDK-Verzeichnisstruktur massiv geändert, kein JDK / JRE mehr
- rt.jar und tools.jar existieren nicht mehr => Module in jmods
- Sichtbarkeitsbeschränkungen => einige interne APIs nicht mehr zugreifbar
- Erweiterung des JDKs mit »endorsed dirs« nicht mehr unterstützt
- Es gibt nun einen Linker, mit dem man spezielle Executables des eigenen Programms erstellen kann.

MODULARISIERUNG DES JDKS




```
.
..
COPYRIGHT
LICENSE
README.html
THIRDPARTYLICENSEREADME-
THIRDPARTYLICENSEREADME.txt
bin
db
include
javafx-src.zip
jre
lib
man
release
src.zip
```



~~jre directory~~
~~tools.jar~~
~~rt.jar~~

```
.
..
COPYRIGHT
LICENSE
README.html
THIRDPARTYLICENSEREADME.txt
bin
bundle
conf
include
jmods
jrt-fs.jar
lib
release
src.zip
```

WAS IST EIN MODUL IM JDK?

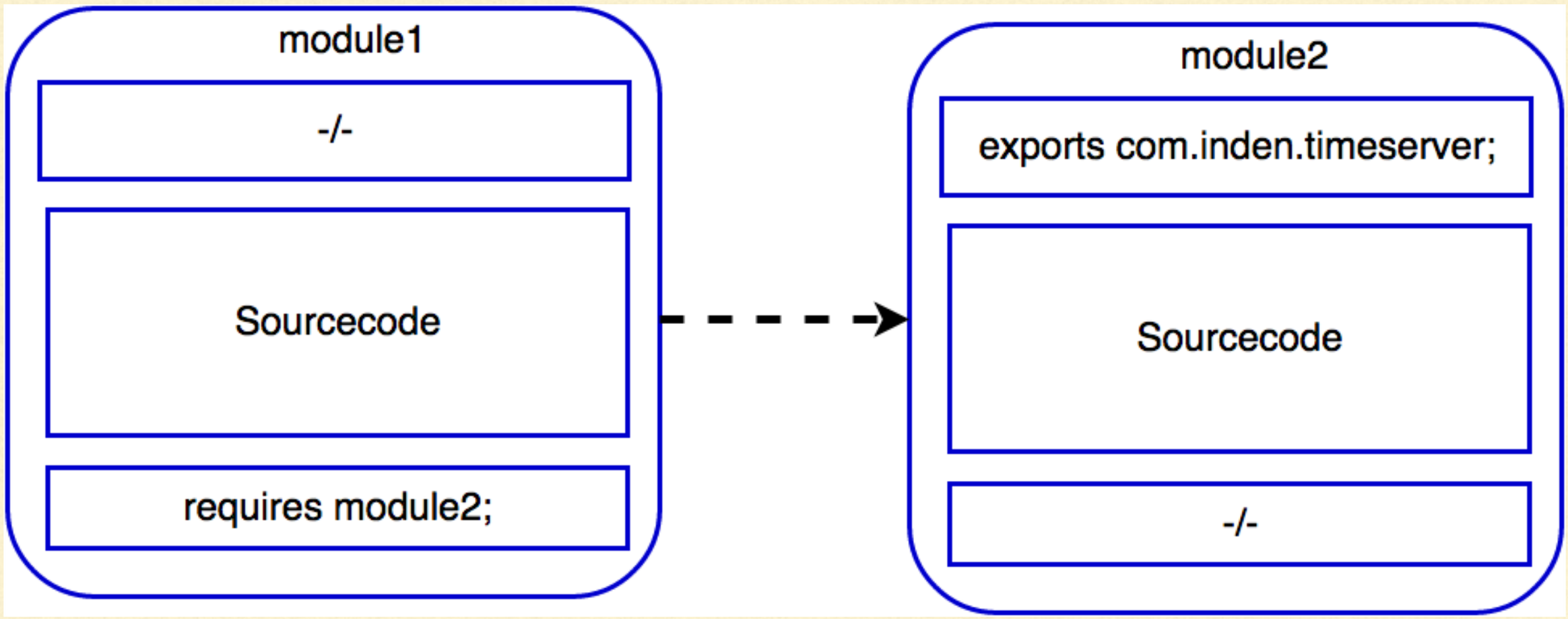
- Eine Menge von Packages und Klassen
- Besitzt einen Moduldeskriptor module-info.java

```
module <ModuleName>
{
    requires <ModuleNameOfRequiredModule>;

    exports <PackageName>;
}
```

- Modulsystem stellt sicher, dass **jede** Abhängigkeit **genau** durch **ein Modul** erfüllt wird und die Abhängigkeiten **azyklisch** sind.

ABHÄNGIGKEITEN BESCHREIBEN



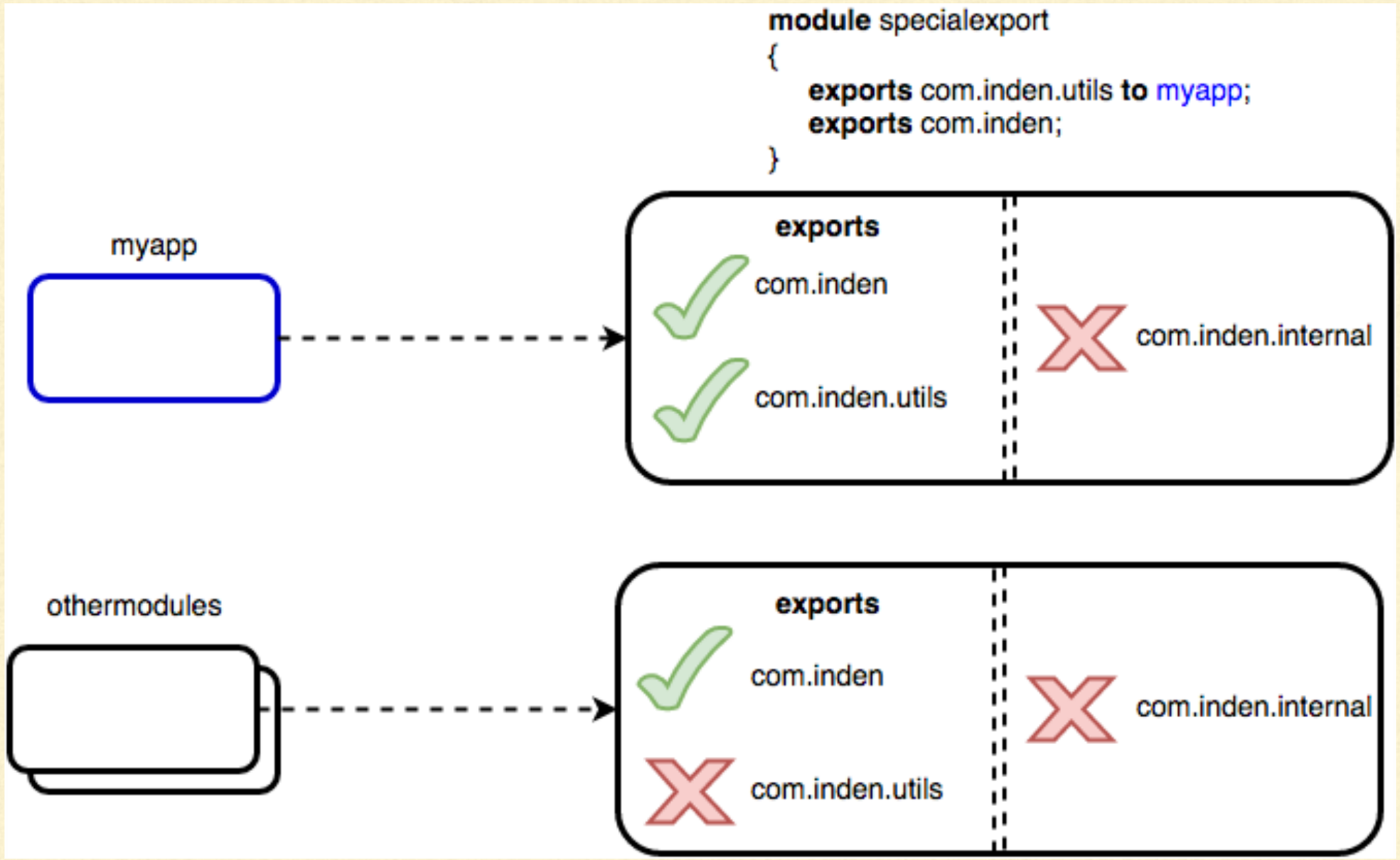

```
module module1
{
    requires module2;
}
```

```
module module2
{
    exports com.inden.timeserver;
}
```

READABILITY / ACCESSIBILITY

- Modul 1 requires Modul 2 => Modul 1 reads Modul 2
- **Readability** ist die Voraussetzung dafür, dass Modul 1 die Typen aus Modul 2 referenzieren kann.
- **Accessibility**: Readability + exports
  Eine Klasse A aus Modul 2 ist nur dann zugreifbar, wenn Modul 1 das korrespondierende Modul 2 liest und Modul 2 zusätzlich das benötigte Package exportiert. => starke Kapselung
- Beides ist die Grundlage für eine verlässliche Konfiguration.

SICHTBARKEITSSTEUERUNG



VERZEICHNISLAYOUT

- Applikation mit mehreren Modulen => common src-Verzeichnis
- Pro Modul wird der Sourcecode in einem Unterverzeichnis mit dem Namen des Moduls abgelegt

```
'- [src]
    |- [de.meinefirma.meinerstesmodul]
    |   |- [de]
    |   |   '- [meinefirma]
    |   |       '- [meinerstesmodul]
    |   |           '- MeineApp.java
    |   '- module-info.java
    '- [de.meinefirma.meinzweitesmodul]
        |- [de]
        |   '- [meinefirma]
        |       '- [meinzweitesmodul]
        |           '- [meinunterpackage]
        |               '- MeineKlasseX.java
        '- module-info.java
```

BEISPIEL MIT

2 MODULEN

2 MODULE

- Schritt 1: Definition der Module
  - Strukturierung im Dateisystem:

```
mkdir -p src/jigsawapp
mkdir -p src/services
```

```
└── src
    ├── jigsawapp
    │   ├── com
    │   │   └── inden
    │   │       └── javaprofi
    │   │           └── MessageExample.java
    │   └── module-info.java
    └── services
        ├── com
        │   └── services
        │       └── MessageService.java
        └── module-info.java
```

  - Moduldeskriptoren:

```
module jigsawapp {
    requires services;
}

module services {
    exports com.services;
}
```

- Schritt 2: Implementierung der Klassen

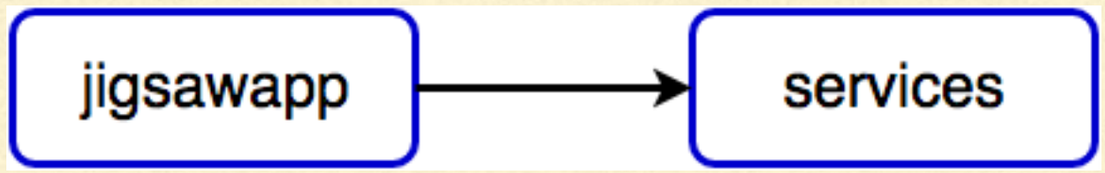
2 MODULE

- Schritt 3: Kompilieren

```
javac -d build \
        src/services/*.java \
        src/services/com/services/*.java

javac -d build \
        src/jigsawapp/*.java \
        src/jigsawapp/com/inden/javaprofi/*.java
```

2 MODULE



- Schritt 3: Kompilieren mit Multi Module Build

```
javac -d build --module-source-path src $(find src -name '*.java')
```

```
build
├── jigsawapp
│   ├── com
│   │   └── inden
│   │       └── javaprofi
│   │           └── MessageExample.class
│   └── module-info.class
└── services
    ├── com
    │   └── services
    │       └── MessageService.class
    └── module-info.class
```

2 MODULE

- Schritt 4: Packaging

```
mkdir lib
jar --create --file lib/services.jar --module-version 1.0 \
    -C build/services .
jar --create --file lib/jigsawapp.jar -C build/jigsawapp .

java -p lib -m jigsawapp/com.inden.javaprofi.MessageExample
```

```
├── lib
│   ├── jigsawapp.jar
│   └── services.jar
```

2 MODULE

- Schritt 4: Packaging mit Executable JAR

```
mkdir lib
jar --create --file lib/services.jar --module-version 1.0
      -C build/services
jar --create --file lib/jigsawapp.jar
      --main-class com.inden.javaprofi.MessageExample
      -C build/jigsawapp .


java -p lib -m jigsawapp/com.inden.javaprofi.MessageExample
java -p lib -m jigsawapp
```

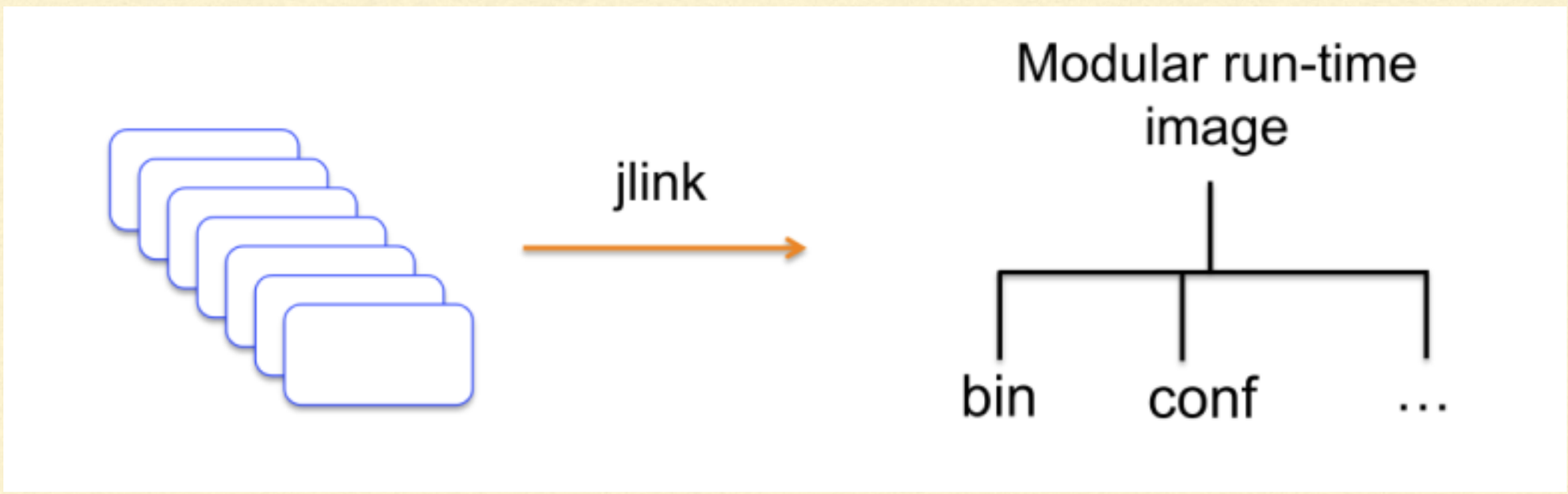
Besonderheiten:

- Linking
- Modulgraph aufbereiten
- Module des JDKs nutzen
- 3rd PARTY Libs verwenden

LINKING / EXECUTABLE JAR

- Executable mit inkludierter Java Runtime erstellen

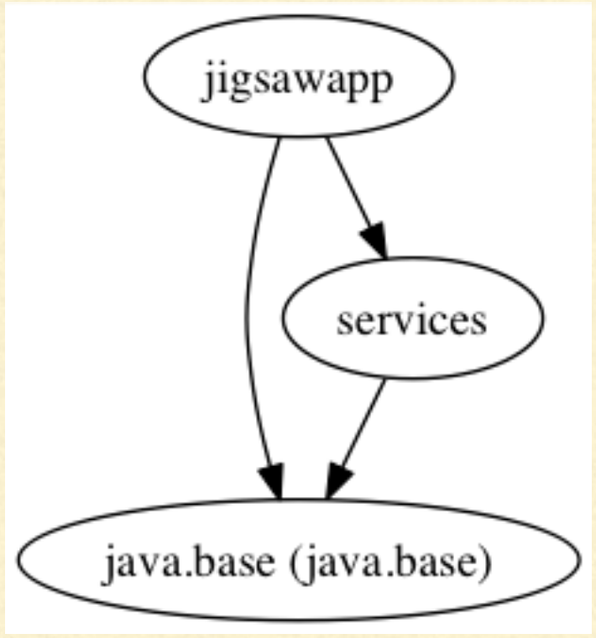
```
jlink --module-path $JAVA_HOME/jmods:lib \
      --add-modules jigsawapp \
      --output moduleexample
```



AUFBEREITUNG MODULGRAPH

- Aufbereitung eines Abhängigkeitsgraphen

```
jdeps --module-path build -dotoutput graphs lib/*.jar
```



MODULE DES JDKS NUTZEN

```
module timeclient
{
    requires javafx.graphics;
    requires javafx.controls;

    requires timeserver;
}

module timeserver
{
    requires java.logging;

    exports com.server;
}
```

```
java –list-modules

java.activation@9-ea
java.annotations.common@9-ea
java.base@9-ea
java.compact1@9-ea
java.compact2@9-ea
java.compact3@9-ea
java.compiler@9-ea
java.corba@9-ea
java.datatransfer@9-ea
java.desktop@9-ea
java.httpclient@9-ea
java.instrument@9-ea
java.jnlp@9-ea
java.logging@9-ea
java.management@9-ea
java.naming@9-ea

…
```

3RD PARTY LIBS ALS MODUL

- Viele Bibliotheken sind noch nicht für Jigsaw ausgelegt! Was nun?
  - Kompatibilitätsmodus alles aus CLASSPATH
  - Migration mit Automatic Modules
- Automatic Modules entstehen, wenn herkömmliches JAR im Module-Path aufgeführt wird. Beispiel guava-19.0.jar

```
module mymodule
{
    requires guava;
}
```

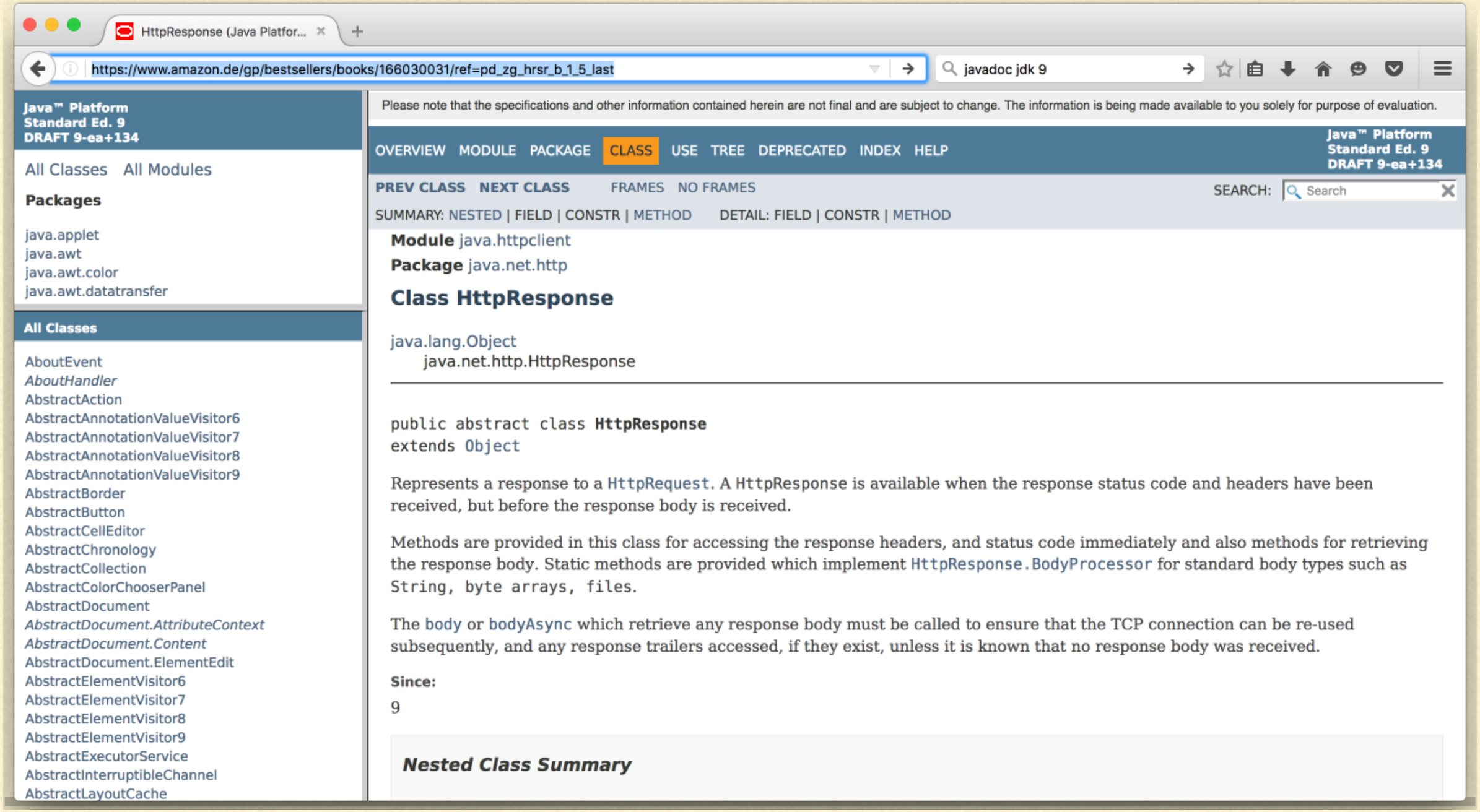
SONSTIGES

SONSTIGES

- Kein Browser-Plugin mehr => Good bye Applets
- G1 wird neuer Standard GC
- Performance-Optimierung „COMPACT-STRINGS“
- UTF-8 Resource Bundles
- Unicode 8 - Support

- diverse weitere Änderungen

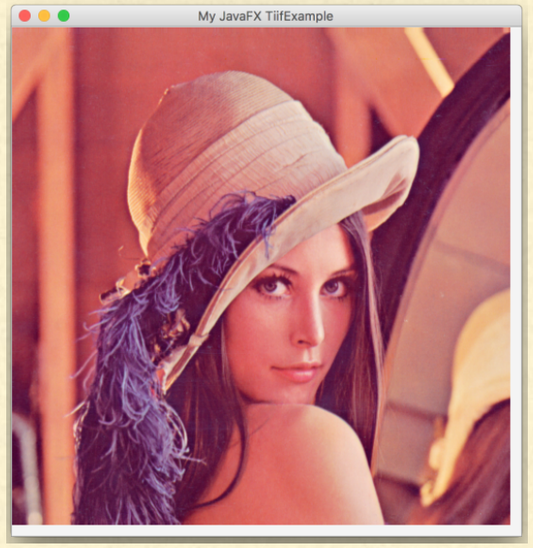
SONSTIGES — HTML 5 JAVADOC



SONSTIGES — GRAFISCHES

- TIFF-Support

```
final InputStream imageInputStream =
                        TiffExample.class.getResourceAsStream("lena_color.tiff");
final BufferedImage image = ImageIO.read(imageInputStream);
```

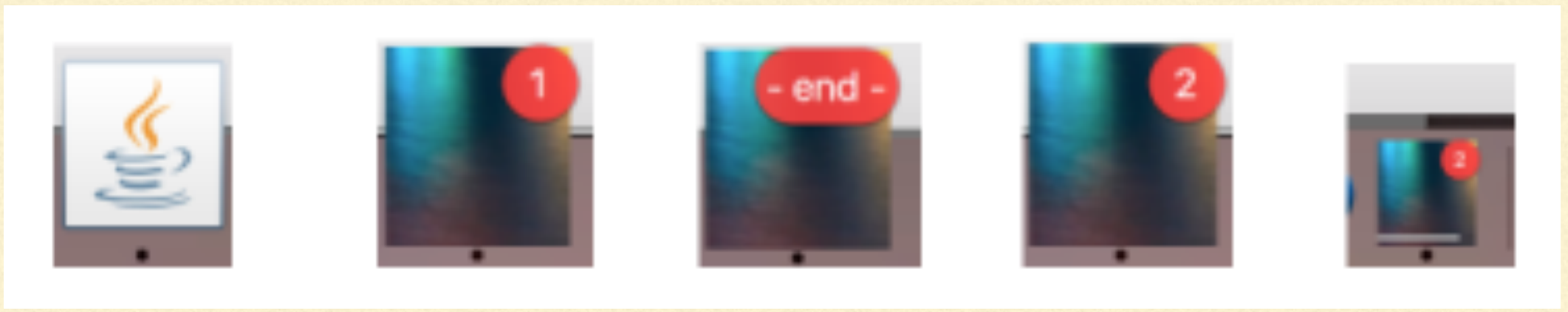


- HiDPI Support mit java.awt.image.MultiResolutionImage

SONSTIGES — GRAFISCHES

- Taskbar-Support

```
if (Taskbar.isTaskbarSupported())
{
    final Taskbar taskbar = Taskbar.getTaskbar();
    taskbar.setIconImage(image);
    taskbar.setIconBadge(text);
    taskbar.setProgressValue(i);
    // ....
}
```



SONSTIGES — VERSIONSNUMMERN

	Old		New	
Release Type	long	short	long	short
Early Access	1.9.0-ea-b19	9-ea	9-ea+19	9-ea
Major	1.9.0-b100	9	9+100	9
Security #1	1.9.0_5-b20	9u5	9.0.1+20	9.0.1
Security #2	1.9.0_11-b12	9u11	9.0.2+12	9.0.2
Minor #1	1.9.0_20-b62	9u20	9.1.2+62	9.1.2
Security #3	1.9.0_25-b15	9u25	9.1.3+15	9.1.3
Security #4	1.9.0_31-b08	9u31	9.1.4+8	9.1.4
Minor #2	1.9.0_40-b45	9u40	9.2.4+45	9.2.4

SONSTIGES — DIVERSES

- MethodHandle
- VarHandle
- Class<T>
- InputStream
- Erweiterungen im Date And Time API
- Erweiterungen in Arrays
- Deprecation der Typen Observer und Observable
- ….

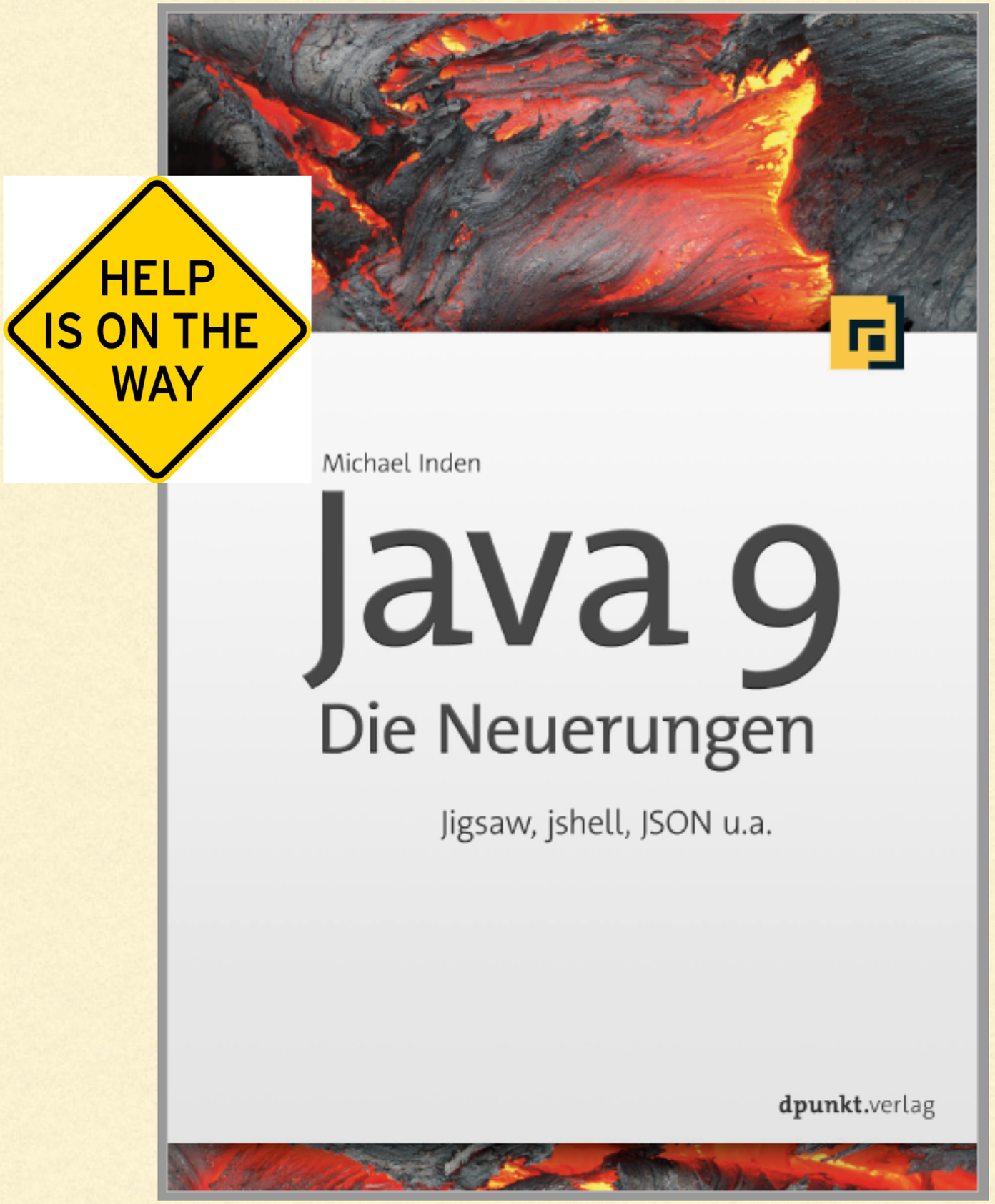
FAZIT

POSITIVES

- Modularisierung mit Project JIGSAW
- Reactive Streams
- HTTP 2
- diverse praktische Erweiterungen in den APIs
- Performance Improvements
- eine paar Dinge aus Project COIN (Sprachsyntax)

MAL WIEDER ;-(((((

- Mal wieder später Sept. 2016 => März 2017 => Juli 2017
- Mal wieder weniger als geplant
  - keine Versionierung bei JIGSAW
  - kein JSON-Support
  - statt Collection-Literals nur Convenience-Methods


HELP
IS ON THE
WAY
Michael Inden
Java 9
Die Neuerungen
Jigsaw, jshell, JSON u.a.
dpunkt.verlag

ÜBUNGEN - Teil 2