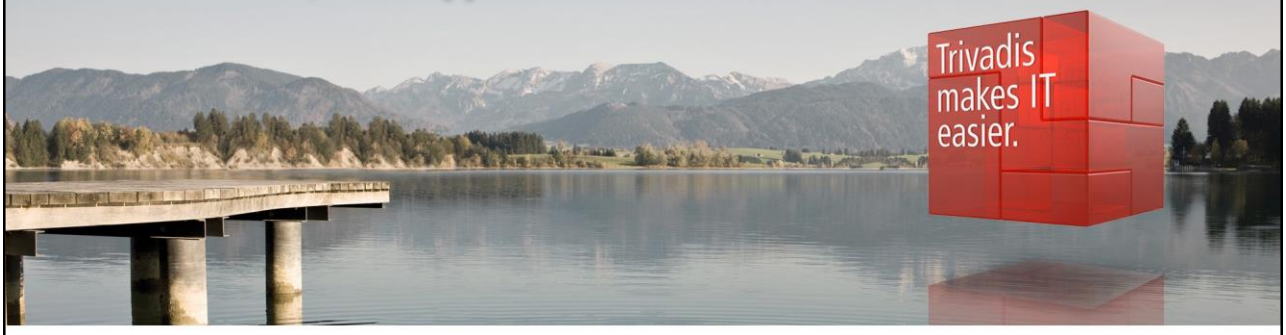
Java 9 ist tot, lang lebe Java 11

Steffen Schäfer, Falk Sippach



Zusammenschluss Trivadis und OIO

Im Mai diesen Jahres haben sich Trivadis und Orientation in Objects (OIO) zusammengeschlossen. Gemeinsam stärken und erweitern wir unser Angebot im Bereich Java und agiler Softwareentwicklung.

Gemeinsam bieten wir ein **zwölfmonatiges Trainee-Programm** an, das Experten für Java- und Webtechnologien ausbildet.

Neu finden Sie im Trivadis **Trainingsangebot** auch Kurse, die von der OIO entwickelt und durchgeführt werden.

OIO Orientation in Objects
trivadis makes IT easier.

Abstract

Seit dem Release von Java 9 hat Oracle ein Rapid-Release-Modell für neue Java-Versionen etabliert. Im halbjährlichen Rhythmus kommen jetzt neue Feature-Releases. Um nicht im Supportwahnsinn zu versinken, wird Oracle nur noch bestimmte Versionen langfristig unterstützen. Das erste dieser LTS-Releases ist Java 11, welches Java 8 als letztes klassisches Release mit langfristiger Unterstützung beerbt hat.

Es ist also an der Zeit, dass wir Java-Entwickler uns einen Überblick über die Neuerungen der vergangenen drei Major-Releases verschaffen. Lässt man das Modulsystem (JPMS/Jigsaw) außer Acht, haben die Java-Versionen 9 bis 11 nämlich noch viele andere, spannende Änderungen mitgebracht. Neben den Sprachänderungen wie "Local Variable Type Inference" möchten wir einen genauen Blick auf die vielen kleinen Erweiterungen der JDK-Klassenbibliothek werfen.

Ihre Sprecher

Steffen Schäfer, Falk Sippach

Trainer, Berater, Entwickler

@sippsack

Orientation in Objects

Einführung

Ist Java 8 schon so alt?

Orientation in Objects

We're counting down to the last free Oracle Java 8 update - what's your plan?

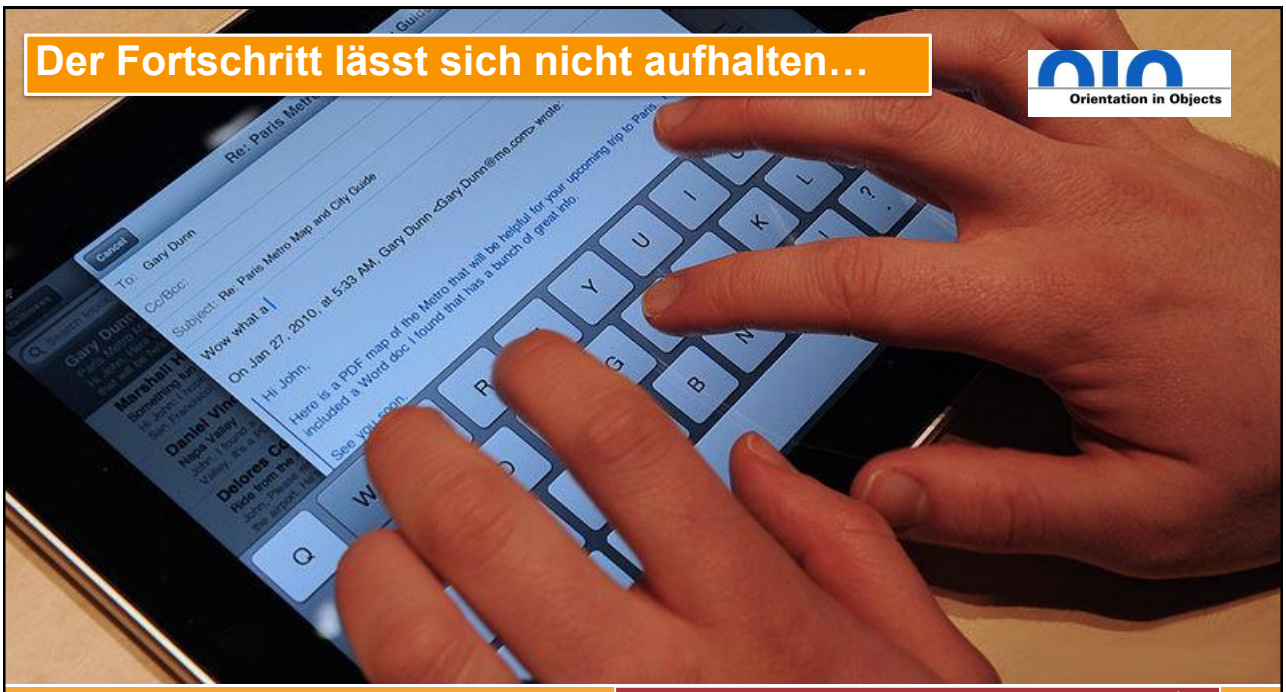
122 days 16 hours 9 min 6 sec

Azul can save your day.

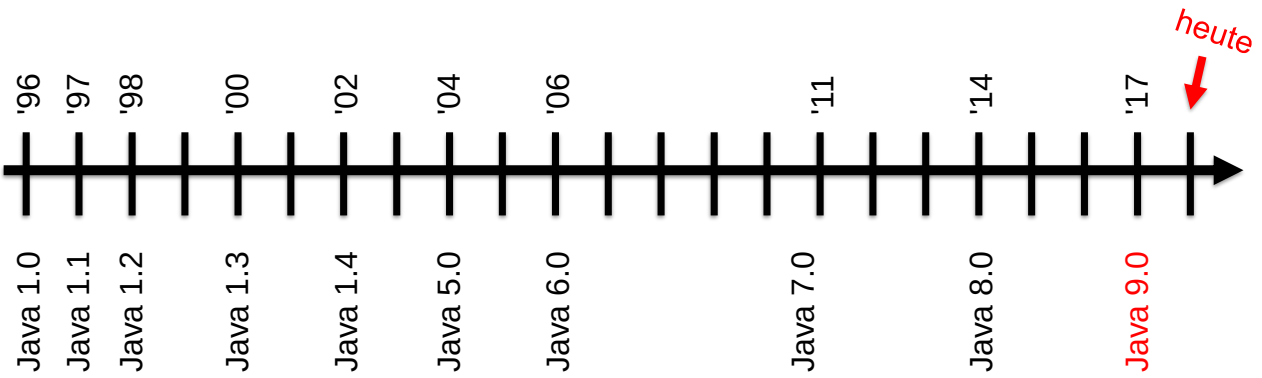
https://www.azul.com/last-free-java-8-download/

Java 9 ist tot?

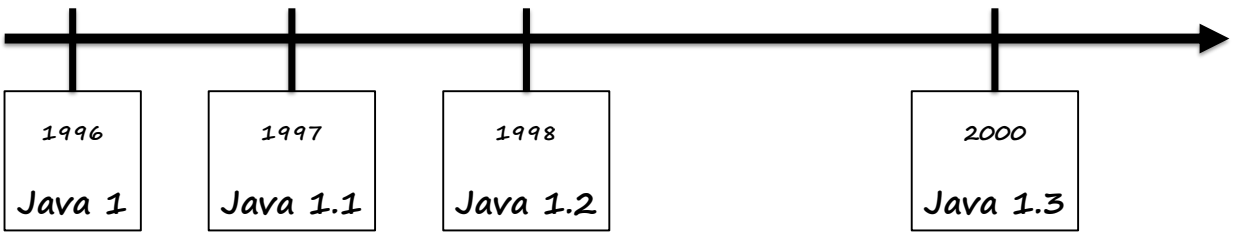
Der Fortschritt lässt sich nicht aufhalten…



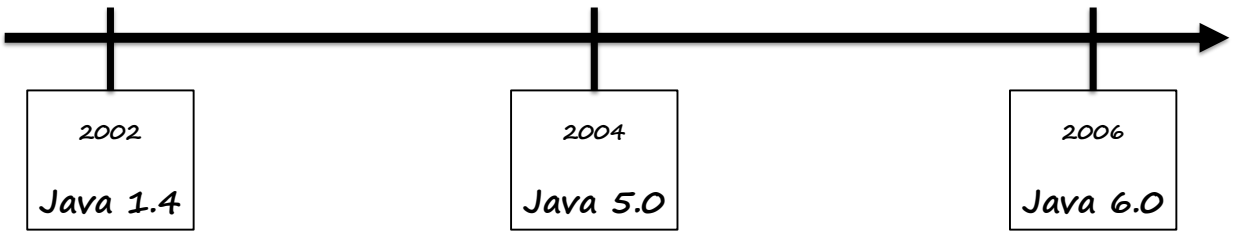
Geschichte



Geschichte

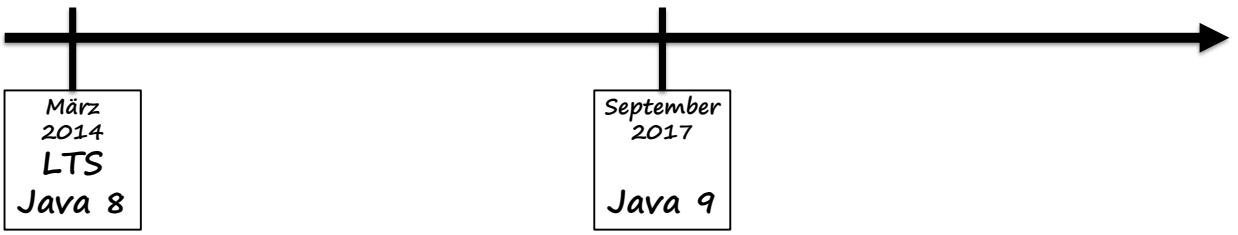


Geschichte

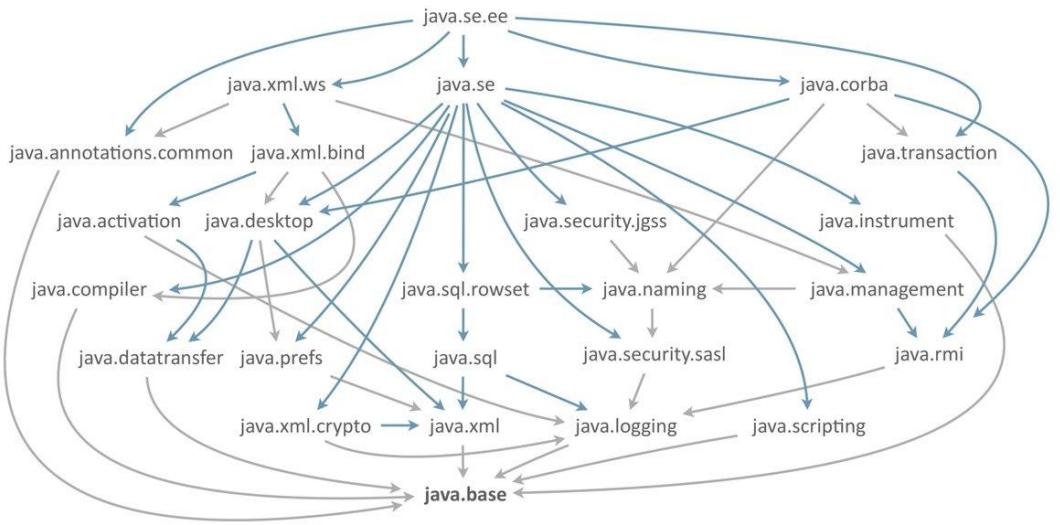


Geschichte

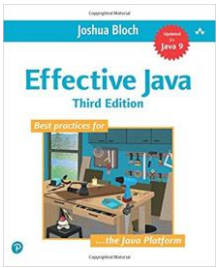
Geschichte



Java 9 nutzt doch sowieso noch keiner …



Modulsystem im Moment noch meiden …

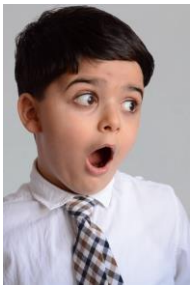


Orientation in Objects

*It is too early to say whether modules **will achieve widespread use outside of the JDK** itself. In the meantime, **it seems best to avoid** them unless you have a compelling need.*

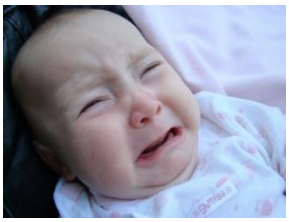
Josh Bloch in "Effective Java: Third Edition"

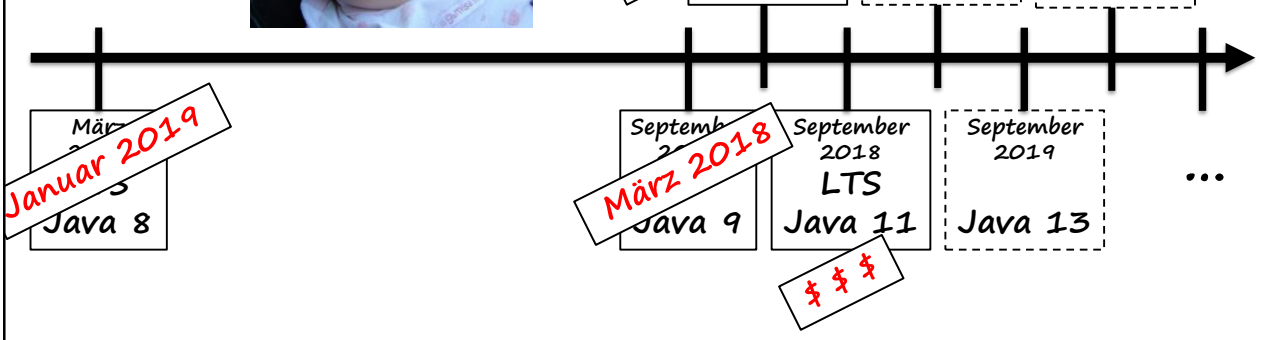
Geschichte



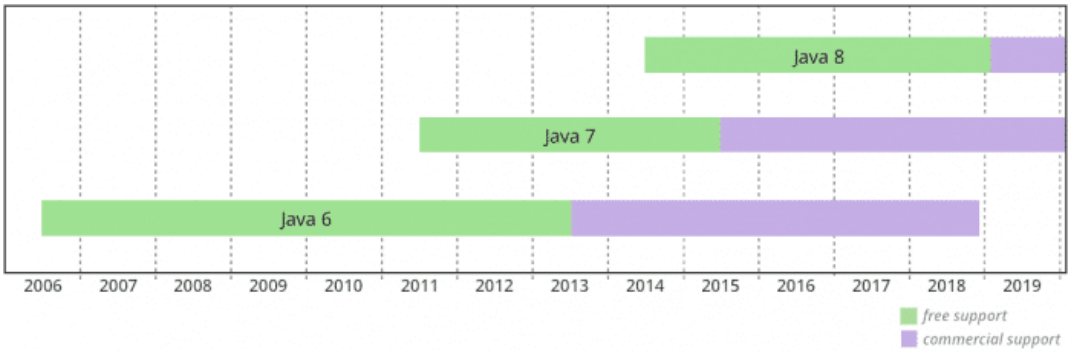
Orientation in Objects

End of Life (Support-Ende) Oracle JDK





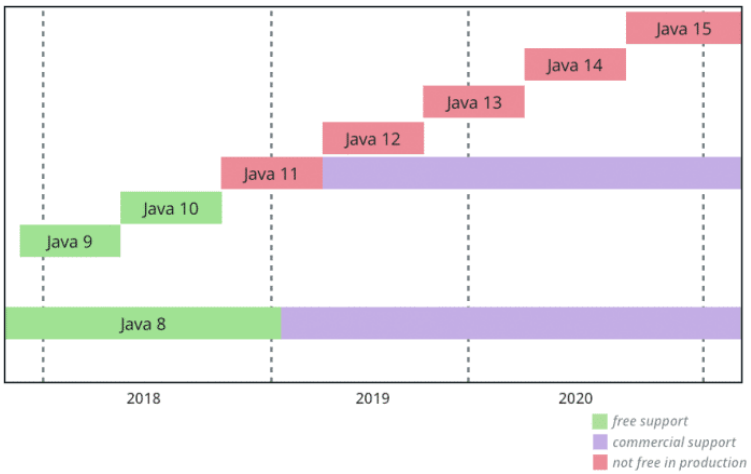
Java hat doch noch nie was gekostet, oder?



https://www.heise.de/developer/artikel/Wird-Java-jetzt-kostenpflichtig-4144533.html

Oracles Free-Lunch is over!

Orientation in Objects



https://www.heise.de/developer/artikel/Wird-Java-jetzt-kostenpflichtig-4144533.html

Orientation in Objects

Neue Features

Java 9 JEPs - 1

Orientation in Objects

102: Process API Updates
110: HTTP 2 Client
143: Improve Contended Locking
158: Unified JVM Logging
165: Compiler Control
193: Variable Handles
197: Segmented Code Cache
199: Smart Java Compilation, Phase Two
200: The Modular JDK
201: Modular Source Code
211: Elide Deprecation Warnings on Import Statements
212: Resolve Lint and Doclint Warnings
213: Milling Project Coin
214: Remove GC Combinations Deprecated in JDK 8
215: Tiered Attribution for javac
216: Process Import Statements Correctly
217: Annotations Pipeline 2.0
219: Datagram Transport Layer Security (DTLS)
220: Modular Run-Time Images
221: Simplified Doclet API
222: jshell: The Java Shell (Read-Eval-Print Loop)
223: New Version-String Scheme
224: HTML5 Javadoc
225: Javadoc Search
226: UTF-8 Property Files
227: Unicode 7.0
228: Add More Diagnostic Commands
229: Create PKCS12 Keystores by Default
231: Remove Launch-Time JRE Version Selection
232: Improve Secure Application Performance
233: Generate Run-Time Compiler Tests Automatically
235: Test Class-File Attributes Generated by javac
236: Parser API for Nashorn
237: Linux/AArch64 Port
238: Multi-Release JAR Files
240: Remove the JVM TI hprof Agent
241: Remove the jhat Tool
243: Java-Level JVM Compiler Interface
244: TLS Application-Layer Protocol Negotiation Extension
245: Validate JVM Command-Line Flag Arguments
246: Leverage CPU Instructions for GHASH and RSA
247: Compile for Older Platform Versions
248: Make G1 the Default Garbage Collector
249: OCSP Stapling for TLS
250: Store Interned Strings in CDS Archives
251: Multi-Resolution Images
252: Use CLDR Locale Data by Default
253: Prepare JavaFX UI Controls & CSS APIs for Modularization
254: Compact Strings
255: Merge Selected Xerces 2.11.0 Updates into JAXP
256: BeanInfo Annotations
257: Update JavaFX/Media to Newer Version of GStreamer
258: HarfBuzz Font-Layout Engine
259: Stack-Walking API

Quelle: http://openjdk.java.net/projects/jdk9/

Java 9 JEPs - 2

Orientation in Objects

260: Encapsulate Most Internal APIs
261: Module System
262: TIFF Image I/O
263: HiDPI Graphics on Windows and Linux
264: Platform Logging API and Service
265: Marlin Graphics Renderer
266: More Concurrency Updates
267: Unicode 8.0
268: XML Catalogs
269: Convenience Factory Methods for Collections
270: Reserved Stack Areas for Critical Sections
271: Unified GC Logging
272: Platform-Specific Desktop Features
273: DRBG-Based SecureRandom Implementations
274: Enhanced Method Handles
275: Modular Java Application Packaging
276: Dynamic Linking of Language-Defined Object Models
277: Enhanced Deprecation
278: Additional Tests for Humongous Objects in G1
279: Improve Test-Failure Troubleshooting
280: Indify String Concatenation
281: HotSpot C++ Unit-Test Framework
282: jlink: The Java Linker
283: Enable GTK 3 on Linux
284: New HotSpot Build System
285: Spin-Wait Hints
287: SHA-3 Hash Algorithms
288: Disable SHA-1 Certificates
289: Deprecate the Applet API
290: Filter Incoming Serialization Data
291: Deprecate the Concurrent Mark Sweep (CMS) Garbage Collector
292: Implement Selected ECMAScript 6 Features in Nashorn
294: Linux/s390x Port
295: Ahead-of-Time Compilation
297: Unified arm32/arm64 Port
298: Remove Demos and Samples
299: Reorganize Documentation

Quelle: http://openjdk.java.net/projects/jdk9/

Java 10 JEPs

Orientation in Objects

286: Local-Variable Type Inference
296: Consolidate the JDK Forest into a Single Repository
304: Garbage-Collector Interface
307: Parallel Full GC for G1
310: Application Class-Data Sharing
312: Thread-Local Handshakes
313: Remove the Native-Header Generation Tool (javah)
314: Additional Unicode Language-Tag Extensions
316: Heap Allocation on Alternative Memory Devices
317: Experimental Java-Based JIT Compiler
319: Root Certificates
322: Time-Based Release Versioning

Quelle: http://openjdk.java.net/projects/jdk/10/

Java 11 JEPs

Orientation in Objects

Features

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

Schedule

2018/06/28	Rampdown Phase One (fork from main line)
2018/07/19	All Tests Run
2018/07/26	Rampdown Phase Two
2018/08/16	Initial Release Candidate
2018/08/30	Final Release Candidate
2018/09/25	General Availability

Quelle: http://openjdk.java.net/projects/jdk/11/

Local-Variable Type Inference

Orientation in Objects

We seek to **improve the developer experience** by **reducing the ceremony** associated with writing Java code, while maintaining Java's commi**tment to static type safety**, by allowing developers to elide the often-unnecessary manifest declaration of local variable types.

JEP 286: Local-Variable Type Inference

Typ Inferenz in Java bis Version 9

Orientation in Objects

- Type Inference bei generisch typisierten Methoden (Java 5)

```
List<String> strings = Arrays.asList("World", "Java 9");
```

- Inferenz von Typinformation bei Lambda-Ausdrücken (Java 8)

```
Function<String, String> helloFunction = s -> "Hello " + s;
```

- Inferenz von Generics via „Diamond Operator“ (Java 7)

```
List<String> helloStrings = new ArrayList<>();
strings.forEach(s -> helloStrings.add(helloFunction.apply(s)));
```

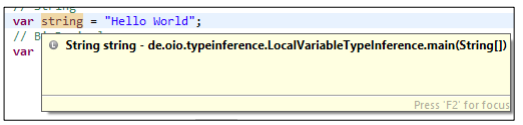
Local Variable Type Inference (JEP 286) - 1

Orientation in Objects

- Neues Schlüsselwort „var“ als Alternative für Typ-Deklaration
  - Nur lokale Variablen, keine Felder oder Methoden-Parameter
  - Compiler inferiert Typinformation aus der Zuweisung
  - Primitiv-Werte, sowie-Objekt-Typen bei Zuweisung möglich

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

- Typinformation im Bytecode vorhanden und durch IDEs nutzbar
  - Keine dynamische Typisierung



Local Variable Type Inference (JEP 286) - 2

Orientation in Objects

- Mit „var“ deklarierte lokale Variablen sind nicht automatisch „final“

```
var zahl = 5;
zahl = 7;
```

- …sie können aber „final“ deklariert werden

```
final var zahl = 5;
```

- Bei Neuzuweisung müssen die Typen passen

```
var zahl = 5;
zahl = 7L;
```

```
LocalVariableTypeInference.java:10: error: incompatible types: possible
lossy conversion from long to int
zahl = 7L;
       ^
1 error
```

Local Variable Type Inference (JEP 286) - 3

Orientation in Objects

- Typ Inferenz auch bei generischen Rückgabewerten möglich

```
// List<String>
var strings = Arrays.asList("World", "Java 10");
```

- Typ Inferenz in Schleifen

```
for (var string : strings) {
    System.out.println("Hello " + string);
}
```

Local Variable Type Inference (JEP 286) - 4

Orientation in Objects

- Verlust von Typinformationen bei Kombination mit Diamond Operator
  - Fallback auf Basistyp des Generics (hier Object)

```
var strings = new ArrayList<>();
strings.add("Hello World");
for (var string : strings) {
  System.out.println(string.replace("World", "Java 10"));
}
```

```
LocalVariableTypeInference.java:63: error: cannot find symbol
System.out.println(string.replace("World", "Java 10"));
                          ^
  symbol:   method replace(String,String)
  location: variable string of type Object
1 error
```

Local Variable Type Inference (JEP 286) - 5

Orientation in Objects

- Typ Inferenz lokaler Typen möglich

```
var myReversibleStringList = new ArrayList<String>() {
  List<String> reverseMe() {
    var reversed = new ArrayList<String>(this);
    Collections.reverse(reversed);
    return reversed;
  }
};
myReversibleStringList.add("World");
myReversibleStringList.add("Hello");

System.out.println(myReversibleStringList.reverseMe());
```

Local Variable Type Inference (JEP 286) - 6

Orientation in Objects

- Typ Inferenz legt konkreten Typ fest

```
var runnable = new Runnable() {
  @Override
  public void run() {}
};
```

- Spätere Zuweisung einer alternative Implementierung nicht möglich

```
runnable = new Runnable() {
  @Override
  public void run() {
  }
};
```

```
LocalVariableTypeInference.java:93: error: incompatible types: <anonymous
Runnable> cannot be converted to <anonymous Runnable>
                        runnable = new Runnable() {
                                   ^
1 error
```

Local Variable Type Inference und Lambdas

Orientation in Objects

- Erweiterung der „Local Variable Type Inference“ mit Java 11
  - „var“ Schlüsselwort für Lambda-Parameter nutzbar
  - Nur Nutzbar, wenn Typ weggelassen werden kann

```
Consumer<String> printer = (var s) -> System.out.println(s);
```

- Mischung von „var“ und deklarierten Typen ist nicht vorgesehen

```
BiConsumer<String, String> printer = (var s1, String s2) ->
System.out.println(s1 + " " + s2);
```

- Mehrwert für Nutzung von Typ-Anotationen

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

Orientation in Objects

Local-Variable Type Inference

Was gab es denn seit Java 9 noch an
Sprach-Neuerungen und **API-Updates**?

Try-with-resources – Neu in Java 9 (JEP 213)

Orientation in Objects

- Try-with-resources mit effectively-final Variable

```
BufferedReader reader =
  new BufferedReader(new InputStreamReader(TryWithResources.class.getResourceAsStream("hello.txt")));
try (reader) {
  System.out.println(reader.readLine());
}
```

Interfaces - Neu in Java 9 (JEP 213)

Orientation in Objects

- Private Methoden in Interfaces jetzt möglich
  - Nicht überschreibbar

```
public interface Hello {
    String makeHello(String name);

    default void sayHello(String name) {
        print(makeHello(name));
    }

    private void print(String string) {
        System.out.println(string);
    }
}

public static void main(String[] args) {
    Hello hello = name -> "Hello " + name + "!";
    hello.sayHello("Dieter Develop");
}
```

Diamond Operator - Neu in Java 9 (JEP 213)

- Doppelte Typinformationen bei anonymen inneren Klassen

```
Consumer<String> stringConsumer = new Consumer<String>() {

    @Override
    public void accept(String string) {
        System.out.println(string);
    }
};
```

- Diamond Operator bei anonymen inneren Klassen

```
Consumer<String> stringConsumer = new Consumer<>() {
    @Override
    public void accept(String string) {
        System.out.println(string);
    }
};
```

UTF-8 Property Resource Bundles (JEP 226)

- Vor Java 9 wurden *.properties Dateien als ResourceBundle stets mit dem Zeichensatz ISO-8859-1 gelesen
  - Eigenschaften können „Unicode Escapes“ enthalten
- Ab Java 9 werden *.properties Dateien als ResourceBundle mit UTF-8 gelesen
  - Kompatibilität für ASCII Zeichen
  - Kompatibilitätsmodus (ISO-8859-1), falls ungültiges Zeichen enthalten
- Zeichensatz explizit wählbar über System Property
  - java.util.PropertyResourceBundle.encoding

Factory Methoden für Collections (JEP 269) - 1

Orientation in Objects

- Erzeugung von List/Set über neue Factories

```
List<String> strings = List.of("a", "b", "c");
Set<String> strings = Set.of("a", "b", "c");
```

- Erzeugte Collections sind unveränderbar

```
List<String> strings = List.of("a", "b", "c");
strings.add("d"); // UnsupportedOperationException
```

- Entspricht damit

```
List<String> strings = Collections.unmodifiableList(Arrays.asList("a",
"b", "c"));
```

Factory Methoden für Collections (JEP 269) - 2

Orientation in Objects

- Erzeugung von Map mit Inhalten bisher

```
Map<String, Integer> values = new HashMap<>();
values.put("a", 1);
values.put("b", 2);
values.put("c", 3);
```

- Erzeugung von Map mit Werten

```
Map<String, Integer> map1 = Map.of("a", 1, "b", 2, "c", 3);
```

- Erzeugte Map ist unveränderlich

```
values.put("d", 4); // UnsupportedOperationException
```

- Erzeugung auf 10 Schlüssel-Wert-Paare begrenzt

```
Map<String, Integer> values = Map.of("a", 1, "b", 2, "c", 3, "d", 4, "e",
5, "f", 6, "g", 7, "h", 8, "i", 9, "j", 10, "k", 11);
```

Factory Methoden für Collections (JEP 269) - 3

Orientation in Objects

- Map.Entry haben wir schon gesehen

```
Map<String, Integer> values = Map.of("a", 1, "b", 2, "c", 3);
for (Map.Entry<String, Integer> entry : values.entrySet()) {
}
```

- Map.Entry Instanzen können jetzt direkt erzeugt werden

```
Map.Entry<String, Integer> entry = Map.entry("a", 1);
```

- Erzeugung von Map mit Werten über Entries

```
Map<String, Integer> map2 = Map.ofEntries(Map.entry("a", 1),
                                          Map.entry("b", 2));
```

Neue Methoden copyOf in Collections - 1

Orientation in Objects

- Defensive Copy ermöglicht unabhängige Kopie einer Collection

```
List<String> strings = new ArrayList<>();
strings.add("a");
List<String> stringsCopy = List.copyOf(strings);
```

- Erzeugte Collection ist nicht änderbar

```
stringsCopy.add("b"); // UnsupportedOperationException
```

- Änderungen an original Collection haben keine Auswirkung

```
strings.add("b");
System.out.println(stringsCopy);
```

- Resultat:

```
[a]
```

Neue Methoden copyOf in Collections - 2

Orientation in Objects

- Typ der kopierten Collection muss nicht gleich sein

```
List<String> strings = new ArrayList<>();
strings.add("a");
strings.add("b");
strings.add("b"); // List kann doppelte Werte aufnehmen

Set<String> stringSet = Set.copyOf(strings);
System.out.println(strings);
System.out.println(stringSet);
```

- Resultat:

```
[a, b, b]
[b, a]
```

Wiederholung: filtern von Stream-Elementen

Orientation in Objects

- Ausfiltern von Elementen eines Streams
  - Intermediate Operation
  - Nur Elemente, für die die Bedingung erfüllt ist, werden weitergegeben

```
Stream.of(1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1)
.filter(x -> x < 6)
.forEach(x -> System.out.print(x + " "));
```

- Resultat:

```
1 2 3 4 5 5 4 3 2 1
```

Neue Stream Methode takeWhile

Orientation in Objects

- takeWhile leitet Elemente weiter, solange eine Bedingung erfüllt ist
- Intermediate Operation
- Keine weiteren Elemente werden weitergeleitet, auch wenn die Bedingung für weitere Elemente wieder erfüllt wird
- Stream-Limitierung mit takeWhile:

```
Stream.of(1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1)
.takeWhile(x -> x < 6)
.forEach(x -> System.out.print(x + " "));
```

- Resultat:

```
1 2 3 4 5
```

Neue Stream Methode dropWhile

Orientation in Objects

- dropWhile verwirft Elemente, solange eine Bedingung erfüllt ist
- Intermediate Operation
- Alle weiteren Elemente werden weitergeleitet, auch wenn die Bedingung für weitere Elemente nicht mehr erfüllt wird
- Stream-Limitierung mit dropWhile:

```
IntStream.of(1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1)
.dropWhile(x -> x < 5)
.forEach(x -> System.out.print(x + " "));
```

- Resultat:

```
5 6 7 6 5 4 3 2 1
```

Neue Variante der Stream Methode iterate - 1

Orientation in Objects

- iterate bisher deklariert als

```
<T> Stream<T> iterate(final T seed, final UnaryOperator<T> f)
```

- Theoretisch unendlicher Stream
- Abbruch z.B. über limit:

```
Stream.iterate(1, x -> x * 2)
.limit(10)
.forEach(x -> System.out.print(x + " "));
```

- Resultat:

```
1 2 4 8 16 32 64 128 256 512
```

Neue Variante der Stream Methode iterate - 2

Orientation in Objects

- Neue Variante von iterate

```
<T> Stream<T> iterate(T seed, Predicate<? super T> hasNext,
UnaryOperator<T> next)
```

- Beispiel:

```
Stream.iterate(1, x -> x < 1000, x -> x * 2).forEach(x ->
System.out.print(x + " "));
```

- Resultat:

```
1 2 4 8 16 32 64 128 256 512
```

- Entspricht:

```
for (int x = 1; x < 1000; x *= 2) {
    System.out.print(x + " ");
}
```

Optional Erweiterungen

Orientation in Objects

- Führt eine der Aktionen aus abhängig von Existenz eines Wertes

```
Optional<String> emptyOptional = Optional.empty();
emptyOptional.ifPresentOrElse(System.out::println, () ->
System.out.println("kein Wert"));
```

- Erstellt neues Optional, falls bestehendes keinen Wert besitzt

```
Optional<String> valueOrA = Optional.empty().or(() -> Optional.of("a"));
```

- Erzeugt Stream mit keinem oder einem Element aus Optional

```
Stream<String> aStream = Optional.of("a").stream();
```

Process API bisher

Orientation in Objects

- ProcessBuilder wurde mit Java 5 hinzugefügt
  - ProcessBuilder zum Starten von Prozessen
  - Process als Referenz auf gestarteten Prozess

```
Process process = new ProcessBuilder("notepad.exe").start();
```

- Interaktion mit dem Prozess über Methoden
  - OutputStream getOutputStream()
  - InputStream getInputStream()
  - InputStream getErrorStream()
  - int waitFor()
  - destroy()

Process API Erweiterungen (JEP 102) - 1

Orientation in Objects

- Neues Interface ProcessHandle
  - Referenzierung beliebiger Prozesse des Systems
- Aktuellen Prozess (JVM) ermitteln

```
ProcessHandle currentProcessHandle = ProcessHandle.current();
```

- Prozess über PID ermitteln

```
Optional<ProcessHandle> notepadProcessHandle = ProcessHandle.of(1337L);
```

- Alle Prozesse zugreifen

```
Stream<ProcessHandle> allProcesses = ProcessHandle.allProcesses();
```

- Ermittlung über Process

```
Process notepadProcess = new ProcessBuilder("notepad.exe").start();
ProcessHandle notepadProcessHandle = notepadProcess.toHandle();
```


Process API Erweiterungen (JEP 102) - 2

Orientation in Objects

- Neues Interface ProcessHandle.Info
  - Informationen laufender Prozesse als „Snapshot“
- Ermittlung über Process

```
Process notepadProcess = new ProcessBuilder("notepad.exe").start();
Info info = notepadProcess.info();
```

- Ermittlung über ProcessHandle

```
ProcessHandle currentProcessHandle = ProcessHandle.current();
Info info = currentProcessHandle.info();
```

Process API Erweiterungen (JEP 102) - 3

Orientation in Objects

- ProcessHandle Methoden
  - long pid()
  - Optional<ProcessHandle> parent()
  - Stream<ProcessHandle> children()
  - Stream<ProcessHandle> descendants()
  - CompletableFuture<ProcessHandle> onExit()
  - boolean destroy()
  - boolean isAlive()
- ProcessHandle.Info
  - Optional<String> command()
  - Optional<String[]> arguments()
  - Optional<Instant> startInstant()
  - Optional<Duration> totalCpuDuration()
  - Optional<String> user()

Process API Erweiterungen (JEP 102) - 4

Orientation in Objects

- Neue Process Methoden
  - long pid()
  - CompletableFuture<Process> onExit()
  - ProcessHandle toHandle()
  - ProcessHandle.Info info()
  - Stream<ProcessHandle> children()
  - Stream<ProcessHandle> descendants()

JShell (JEP 222) - 1

Orientation in Objects

- JShell ist Read-Evaluate-Print Loop (REPL)
  - Einfacher Test von API Methoden
  - Erstellung von Skripten
- Interaktiver Kommandozeilen Interpreter
  - Schnipsel von Java Quelltext kann eingegeben und direkt ausgeführt werden
  - Ergebnisse sind direkt über generierte Variablen verfügbar

```
$ jshell
|  Welcome to JShell -- Version 9.0.4
|  For an introduction type: /help intro

jshell> "Hello Java 9"
$1 ==> "Hello Java 9"

jshell> System.out.println($1)
Hello Java 9

jshell> /exit
|  Goodbye
```

JShell (JEP 222) - 2

Orientation in Objects

- Alle Ergebnisse werden in nummerierte Variablen mit „$“ als Präfix gespeichert
  - Deklarierte Primitivwerte und Strings
  - Rückgabewerte von aufgerufenen Methoden

```
jshell> "Hello JShell"
$1 ==> "Hello JShell"

jshell> $1.replace("JShell", "Java 9")
$2 ==> "Hello Java 9"
```

- Deklaration von Variablen mit Namen und Typ möglich

```
jshell> String hello="Hello JShell"
hello ==> "Hello JShell"

jshell> System.out.println(hello)
Hello JShell
```

JShell (JEP 222) - 3

Orientation in Objects

- Definition und Nutzung von Methoden
  - Mehrzeilige Eingaben durch Shift+Enter

```
jshell> String flip(String s) {
   ...> StringBuilder flippedString = new StringBuilder();
   ...> for (int i = (s.length() - 1); i >= 0; i--) {
   ...> flippedString.append(s.charAt(i));
   ...> }
   ...> return flippedString.toString();
   ...> }
|  created method flip(String)

jshell> System.out.println(flip("!dlroW olleH"))
Hello World!
```

JShell (JEP 222) - 4

Orientation in Objects

- Speicherung einer interaktiven Session in eine Datei

```
jshell> "Hello JShell"
$1 ==> "Hello JShell"

jshell> System.out.println($1)
Hello JShell

jshell> /save hello.jsh
```

- Öffnen eines Skripts aus der JShell

```
jshell> /open hello.jsh
Hello JShell
```

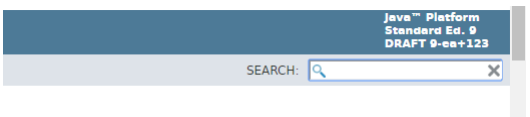
- Öffnen eines Skripts von der Kommandozeile

```
$ jshell hello.jsh
Hello JShell
```

JavaDocs

Orientation in Objects

- JEP 224: HTML5 Javadoc
  - Enhance the javadoc tool to generate HTML5 markup.
- JEP 225: Javadoc Search
  - Add a search box to generated API documentation that can be used to search for program elements and tagged words and phrases within the documentation. The search box will appear in the header of all pages that are displayed in the main right hand frame.



HTTP Client (JEP 110, 321)

Orientation in Objects

- Neuer HTTP Client
  - Incubating/Feedback-Loops in Java 9/10
  - Stabile Version und Standardisierung ab Java 11
- Unterstützt
  - HTTP/1.1 und HTTP/2
  - WebSockets
  - HTTP/2 Server Push
  - Synchrone und Asynchrone Aufrufe
  - reactive-streams

Garbage Collection und Memory Management

Orientation in Objects

- Compact Strings (JEP 254)
  - Java Strings sind in UTF-16 kodiert (2 Byte pro Zeichen)
  - Strings mit ausschließlich westlichen Zeichen werden implizit als ISO-8859-1 kodiert (1 Byte pro Zeichen)
- Garbage Collector G1 ist jetzt Standard (JEP 248)
  - Minimierung von GC Pausen
  - Deduplikation von Strings
- Parallelisierung von Full GC beim G1 (JEP 307)
  - Minimierung von „Stop the World“ Pausen

Was gab es sonst noch Neues seit Java 9?

Orientation in Objects

- Modulsystem JPMS
- Flow API (reaktive Streams API)
- Klassifizierung von Deprecations
- diverse API-Änderung
- …

Orientation in Objects

Fazit

Orientation in Objects

These:

Java 11 finalisiert angefangene Arbeiten aus Java 9 und 10, damit es als LTS Release für die nächsten 3 Jahre gut da steht.

Java 11

Orientation in Objects

- 25.09.2018 General Availability
- bescheidene Sprach- und API-Änderungen
  - 323: Local-Variable Syntax for Lambda Parameters
  - 321: HTTP Client (Standard)
  - 330: Launch Single-File Source-Code Programs
- Aufräumarbeiten
  - 320: Remove the Java EE and CORBA Modules
  - Client-Technologien (JavaFX, Web Start, Applets) entfernt

Was fliegt raus? Was wird deprecated?

Orientation in Objects

- Deprecation in JDK 9 und JDK 10
- Entfernt in JDK 11
  - 320: Remove the Java EE and CORBA Modules
  - Client-Technologien (JavaFX, Web Start, Applets) entfernt
- Deprecation in JDK 11
  - 335: Deprecate the Nashorn JavaScript Engine

Lizenzpolitik

Orientation in Objects

- halbjährliche Releases seit Java 10 im März und September (Java 11 = 18.9, Java 12 = 19.3, …)
- Oracle JDK ist binärkompatibel zu OpenJDK
- End of Life für JDK 8, 9, 10 bereits erreicht oder folgt in Kürze
- Oracle JDK 11 ist das neue LTS (Long Term Support) Release
- Oracle JDK 11 ist in Produktion nicht mehr kostenlos
- Updates für OpenJDK 11 und höher gibt es nur noch für ein halbes Jahr

Preise

Orientation in Objects

Anzahl Prozessoren	Monatlicher Preis pro Prozessor
1-99	$25.00
100-249	$23.75
250-499	$22.50
500-999	$20.00
1.000-2.999	$17.50
3.000-9.999	$15.00
10.000-19.999	$12.50

Was macht die Konkurrenz?

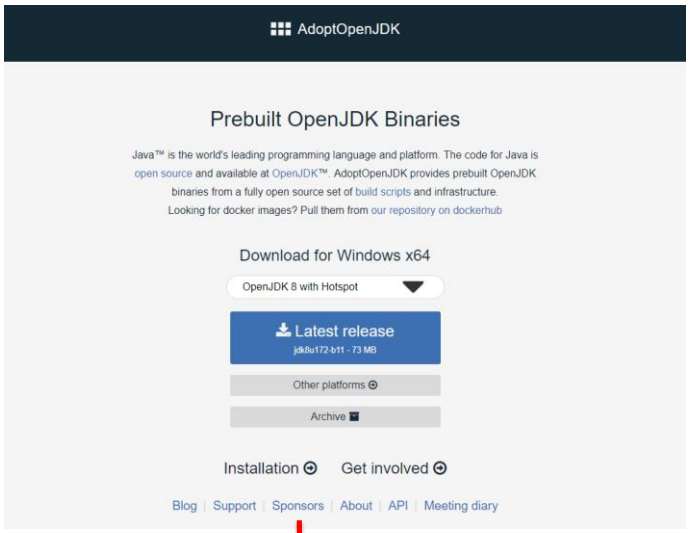
Orientation in Objects

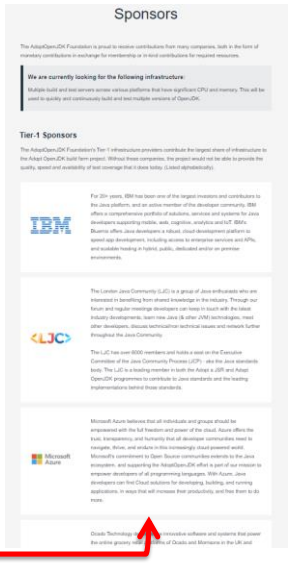
- Azul Zulu
- IBM J9 und Eclipse OpenJ9
- AdoptOpenJDK

LTS Release der Community?

Orientation in Objects

https://twitter.com/stevewallin/status/964515481003667456





https://adoptopenjdk.net/sponsors.html

Based upon this roadmap, and starting with Java 8:

	First Availability	End of Availability
Java 8 (LTS)	March 2014	Sept 2022 [Note 1]
Java 9	Sept 2017	March 2018
Java 10	March 2018	Sept 2018
Java 11 (LTS)	Sept 2018	Sept 2022

Notes:
[1] Java 8 will be maintained until September 2022 to allow applications time to transition from Java 8 LTS to the new technology in Java 11 LTS onwards.

https://adoptopenjdk.net/support.html

Orientation in Objects

Support Levels

As a community of open source developers, our commitment is to triage any issues raised and champion them in the appropriate source code project. Of course, if the problem arises from the way we build and test the code we can fix that directly -- but other bugs will be fixed on a "best effort" basis by the correct source project community. If you are looking for higher levels of assurance you should contact commercial companies offering support on these binaries.

https://adoptopenjdk.net/support.html

Java 12 JEPs

Orientation in Objects

Features

JEPs targeted to JDK 12, so far

- 325: Switch Expressions (Preview)
- 326: Raw String Literals (Preview)

Quelle: http://openjdk.java.net/projects/jdk/12/

Switch expressions (JEP 325)

Orientation in Objects

```
/**
 * Demonstrate traditional s
 * local variable.
 */
public static void demonstra
{
   out.println("Traditional
   final int integer = 3;
   String numericString;
   switch (integer)
   {
      case 1 :
         numericString = "or
         break;
      case 2 :
         numericString = "tu
         break;
      case 3:
         numericString = "th
         break;
      default:
         numericString = "N/
   }
   out.println("\t" + intege
}
```

Traditionell

```
/**
 * Demonstrate switch expression using
 */
public static void demonstrateSwitchEx
{
   final int integer = 1;
   out.println("Switch Expression with
   final String numericString =
      switch (integer)
      {
         case 1 :
            break "uno";
         case 2 :
            break "dos";
         case 3 :
            break "tres";
         default :
            break "N/A";
      };
   out.println("\t" + integer + " ==> " + numericString);
}
```

Expression mit Breaks

```
/**
 * Demonstrate switch expressions using "arrow" syntax.
 */
public static void demonstrateSwitchExpressionWithArrows()
{
   final int integer = 4;
   out.println("Switch Expression with Arrows:");
   final String numericString =
      switch (integer)
      {
         case 1 -> "uno";
         case 2 -> "dos";
         case 3 -> "tres";
         case 4 -> "quatro";
         default -> "N/A";
      };
   out.println("\t" + integer + " ==> " + numericString);
}
```

Expression mit Arrow

https://dzone.com/articles/jdk-12-switch-statementsexpressions-in-action

Raw String Literal (JEP 326)

Orientation in Objects

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

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

```
String html = "<html>\n" " <body>\n" +
       " <p>Hello World.</p>\n" +
       " </body>\n" + "</html>\n";
```

```
String html = `<html> <body>
<p>Hello World.</p> </body>
</html>`;
```

"A raw string literal can span multiple lines of source code and does not interpret escape sequences, such as \n, or Unicode escapes, of the form \uXXXX."

Zukunft

Orientation in Objects

- Java 12 bereits in Arbeit (Termin: März 2019)
- neue Ideen im Inkubator:
  - Pattern Matching
  - Value Types
  - User-Mode Threads

Amber
Valhalla
Loom

The goal of Project _____ is to **explore and incubate** (smaller, productivity-oriented) **Java language and VM features** that have been accepted as **candidate JEPs** under the **OpenJDK JEP process**.

Zukunft

Orientation in Objects

Vielen Dank. Fragen?

Steffen Schäfer
Falk Sippach

+49-621-718390

@oio_braintime **http://blog.oio.de**

Orientation in Objects

trivadis
makes IT easier.