

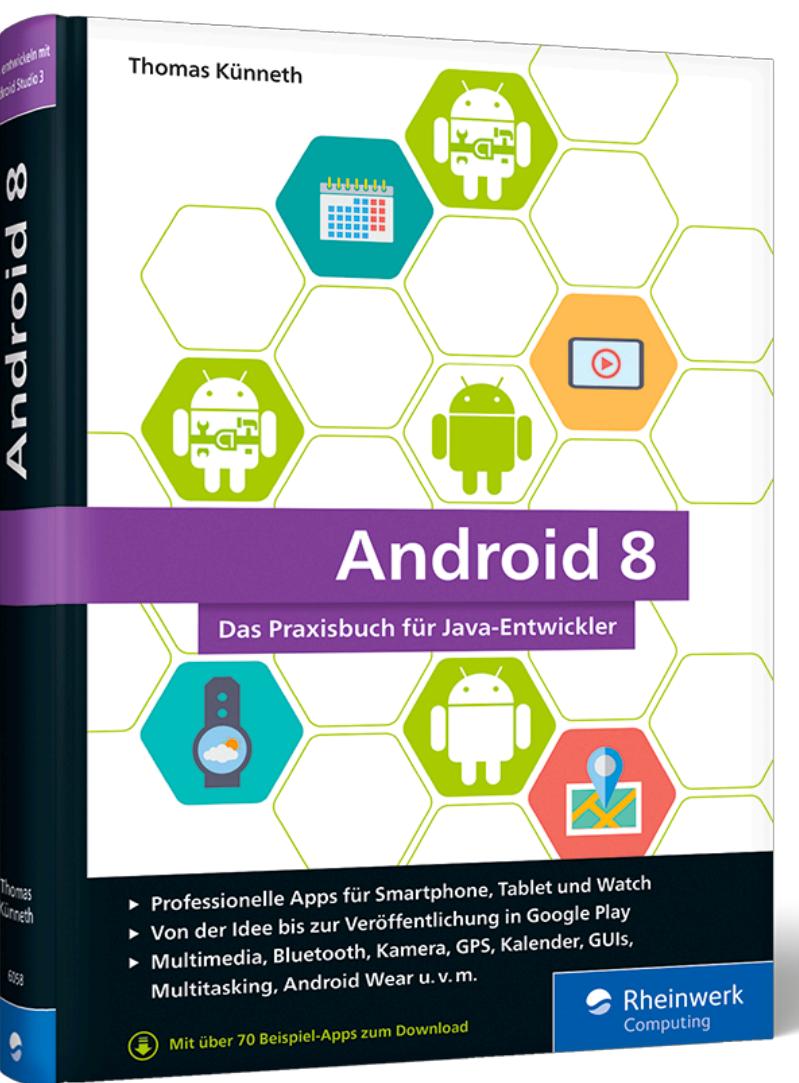
10 vor 11 oder 5 vor 12

Wieviel Frontend steckt noch in Java?

Thomas Künneth
MATHEMA Software GmbH

- Frühjahr 2018: Oracle schreckt Community mit **Java Client Roadmap Update** auf
- Mit Java 11 fallen altgediente Client-Features weg
- Weitere werden für veraltet erklärt
- Sorge, ab 2019 für Updates Geld ausgeben zu müssen

- Wieviel Frontend steckt noch in Java?
- Was ist schon weggefallen?
- Was wird noch wegfallen?
- Wie gehe ich mit der Situation um?



<https://www.mathema.de/unternehmen/mitarbeiter/thomas-kuenneth>

thomas.kuenneth@mathema.de



Wichtige Client-bezogene Technologien bis Java SE 11

- App-Verteilung (Applets und Web Start)
- UI (AWT, Swing, JavaFX)
- Java Control Panel and Auto Update
- Browserintegration

Wichtige Client-bezogene Technologien *ab Java SE 11*

- ~~App-Verteilung (Applets und Web Start)~~
- UI (AWT, Swing, ~~JavaFX~~)
- ~~Java Control Panel and Auto Update~~
- ~~Browserintegration~~

Wie gehe ich mit dem Feature-Schwund um?

- So lange wie möglich auf Java SE 8 bleiben
- Oder dorthin migrieren
- Auf aktuellere Java-Versionen wechseln und Apps entsprechend anpassen

Welches Vorgehen ist das richtige?

- Abhängig von der Art der App und Benutzerkreis
- Apps für Endanwender sollten App Store-fähig sein
- Bei Enterprise Apps Kontinuität und Kosten abwägen
 - Wie kommen die Anwendungen auf den Arbeitsplatz-Rechner?
 - Wie starten Mitarbeiter die Anwendung?
 - Ggf. Schulungsbedarf

Was bedeutet „Bei Java SE 8 bleiben“?

- Java SE 8 kommt als...
 - Oracle JDK
 - OpenJDK
- Auswahl hat ggf. Auswirkungen auf **Kompatibilität, Lizenzbedingungen und Kosten**

Oracle Java SE 8

- Öffentliche Verfügbarkeit nur bis Januar 2019
- Privatanwender: Updates bis Ende 2020 via Auto-Update
- Geänderte Lizenz beachten (z. B. bzgl. Weitergabe oder Bündelung mit Apps)
- Bei kommerzieller Nutzung (in Produktion) fallen Kosten an

OpenJDK 8

- Komfortable Lizenz: GPL mit Classpath Exception
- Mehrere Firmen bieten Distributionen an
- Keine 100%-Kopie des Oracle JDK
 - Nicht alle Bestandteile des Oracle JDK konnten open sourced werden
 - Z. B. kein Web Start (ggf. IcedTea-Web ausprobieren)

Zwischenstand

- Auch beim Verbleib auf Java SE 8 Aufwand und Kosten
 - Ggf. Migration hin zu Java SE 8
 - Wechsel auf OpenJDK: Test und ggf. Anpassungen
 - Oracle JDK: Ggf. Abschluss eines Support-Abos
 - Auch bei anderen LTS-Anbietern ggf. Kosten
 - Nur Option auf Zeit

Applets und Web Start

bei Endanwendern

- Applets spielen schon lange keine Rolle mehr
- Web Start war cool für Demos (aber nie echte Relevanz)

Nicht beliebt bei Endanwendern

- Unterstellung: Einfallstor für Malware
- Hässliches Kontrollfeld
- Liebloser Update-Prozess
- Schlampige Integration in das Wirtssystem

Auf dem Enterprise-Client

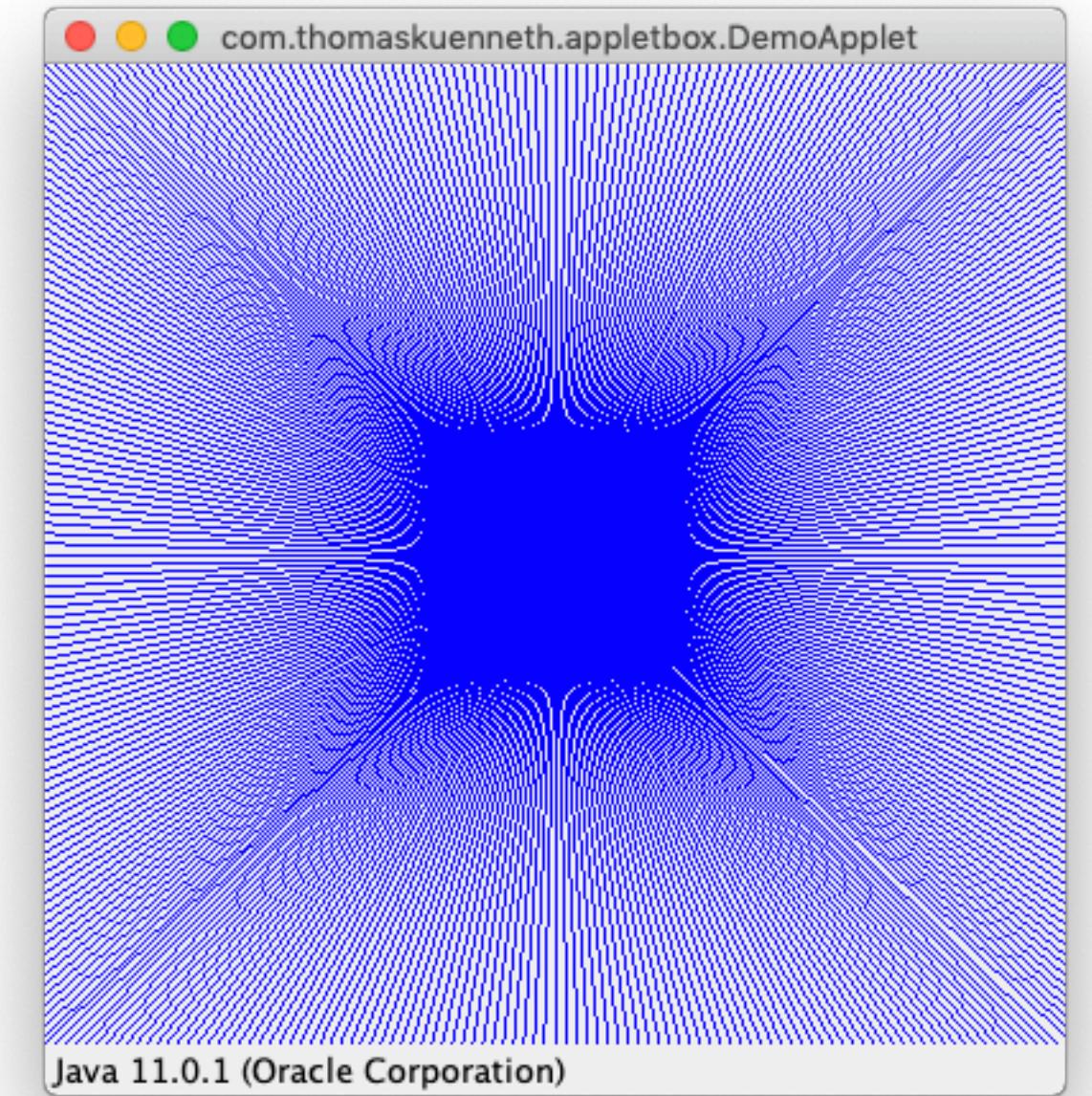
- JREs wurden zentral ausgerollt
- Web Start und Applets häufig Bestandteil der Client-Strategie

Empfehlung

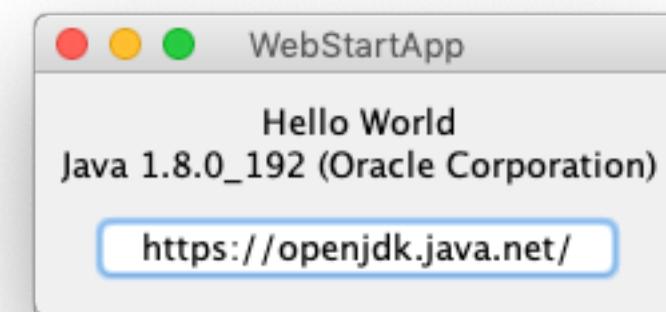
- Nur bei Enterprise-Web Start-Anwendungen und Applets Verbleib auf Java SE 8 in Erwägung ziehen
- Bei Applets Wegbrechen der Browser-Unterstützung beachten
- Im kommerziellen Umfeld Web Start-Unterstützung durch Oracle bis Frühjahr 2025, für Endanwender bis Ende 2020
- Wenn möglich Ablösung und Migration auf Java SE 11

Applets

- In Java SE 11 kein Browser-Plugin und kein appletviewer
- Applet-bezogene Klassen und Interfaces aber weiterhin vorhanden
- Einfacher „Applet Viewer“ schnell fertig



Web Start



- `javaws` nicht in Java SE 11 enthalten
- Üblicherweise lassen sich Web Start-Apps aber über die Kommandozeile starten (`java -jar ...`)

An die JNLP API denken

- Zugriff auf das lokale Dateisystem mit FileOpenService und FileSaveService
- Mehrere Instanzen einer App mit dem SingleInstanceState koordinieren
- BasicService hilft beim Anzeigen von Webseiten

Einfach anpassen

- Je nach Anzahl der betroffenen Apps entweder...
 - Anwendungen direkt anpassen
 - Bibliothek mit möglichst einfach nachgebauten Klassen verwenden
 - Hinweis für Java 9/10: Projekt entweder um module-info.java erweitern oder beim Übersetzen --add-modules java.jnlp

JavaFX

- In Java SE 11 nicht enthalten
- Aktive Entwicklung durch OpenJFX Open Source Projekt und engagierter Community
- JavaFX11 Builds unter <https://openjfx.io/>
 - JavaFX SDK for Windows, Linux und macOS
 - JavaFX jmods for Windows, Linux und macOS

Integration



- Durch Referenzieren der JavaFX jmods
- Durch Hinzufügen der JavaFX-Artefakte in die POM

Paketieren von Apps

- javafxpackager (später javapackager)
- Konnte „self contained“ Apps und native Installationsdateien erstellen
- Nicht in Java SE 11 enthalten
- Nicht im JavaFX 11 SDK enthalten

Geplanter Nachfolger (JEP 343)

- Soll App und Java Runtime Image als Input erhalten
- Wird ein modulares Laufzeit-Image und optional eine native Installationsdatei erstellen
- Seit Ende Oktober Code in Review
- Kandidat für Java SE 12

Was tun?

- `javapackager` aus JDK 10 verwenden
- Auf Java SE 11 hoffen
- „Roll your own“ (JEP 220: Modular Runtime Images)

Modular Runtime Images

- Werden seit Java SE 9 mit dem Java Linker (`jlink`) erzeugt
- Enthalten eine an die Anforderungen der App angepasste Untermenge der JRE
- Voraussetzung: Module mit `module-info.java`
- `jlink`, `jdeps`, `jmod` sowie ggf. plattformspezifische Tools für den Bau nativer Installationsdateien

War's das?

- Keine CORBA- und Java EE-Module mehr ([JEP 320](#))
 - Common Annotations, JAX-WS, JAXB, JTA, JAF als Maven-Artefakte [referenzierbar](#)
 - Bzgl. CORBA mehr Arbeit nötig
 - Pack200 Tools und API veraltet
 - Viel vermeintlicher „Kleinkram“ fällt weg

com.sun.awt.AWTUtilities

The screenshot shows a web browser window with the URL https://docs.oracle.com/javase/tutorial/uiswing/misc/trans_shaped_windows.html. The page title is "Java SE 6 Update 10 API". A search bar at the top right contains the text "com.sun.awt.AWTUtilities". The main content discusses the move of translucent and shaped window functionality from the private `com.sun.awt.AWTUtilities` class to the public `AWT` package in JDK 7. It includes a table mapping private methods to their public equivalents.

Changing the public API in an update release is not allowed, so when the translucent and shaped windows capability was added to the Java SE 6 Update 10 release, it was implemented in the private `com.sun.awt.AWTUtilities` class. For the JDK 7 release, this functionality was moved to the public AWT package. The following table shows how the private methods map to the public methods.

Method in Java SE 6 Update 10	JDK 7 Equivalent
<code>AWTUtilities.isTranslucencySupported(Translucency)</code>	<code>GraphicsDevice.isWindowTranslucencySupported(WindowTranslucency)</code>
<code>AWTUtilities.isTranslucencyCapable(GraphicsConfiguration)</code>	<code>GraphicsConfiguration.isTranslucencyCapable()</code>
<code>AWTUtilities.setWindowOpacity(Window, float)</code>	<code>Window.setOpacity(float)</code>
<code>AWTUtilities.setWindowShape(Window, Shape)</code>	<code>Window.setShape(Shape)</code>
<code>AWTUtilities.setWindowOpaque(boolean)</code>	<code>Window.setBackground(Color) Passing new Color(0,0,0,alpha) to this method, where alpha is less than 255, installs per-pixel translucency.</code>

« Previous • Trail • Next »

About Oracle | Contact Us | Legal Notices | Terms of Use | Your Privacy Rights
Copyright © 1995, 2017 Oracle and/or its affiliates. All rights reserved.

Bald keine Nashörner mehr

- Ab Java SE 11 Script Engine Nashorn und `jjS` veraltet
[\(JEP 335\)](#)
- Seit Java SE 8 an Bord
- Nachfolger der Script Engine Mozilla Rhino (in Java SE 6 und 7 enthalten)

```
[TommisMacBookPro:dist thomas$ jjs -v
nashorn 11.0.1
Warning: The jjs tool is planned to be removed from a future JDK release
[jjs> var a = "Hello world";
[jjs> print(a);
Hello world
[jjs> quit();
TommisMacBookPro:dist thomas$
```

The screenshot shows the Apache NetBeans IDE 9.0 interface. The 'Projects' panel on the left lists several demo projects like AppletBox, FontDemo, JavaFXDemo, JNLPAPI, and ScriptingDemo. The 'ScriptingDemo' project is selected, and its 'Source Packages' node contains a single file named 'ScriptingDemo.java'. The code in this file is as follows:

```
1 package com.thomaskuenneth.scriptingdemo;
2
3 import java.util.List;
4 import java.util.logging.Level;
5 import java.util.logging.Logger;
6 import javax.script.ScriptEngine;
7 import javax.script.ScriptEngineFactory;
8 import javax.script.ScriptEngineManager;
9 import javax.script.ScriptException;
10
11 /**
12 * @author Thomas Kuenneth
13 */
14 public class ScriptingDemo {
15
16     private static final String ECMA_SCRIPT = "ECMAScript";
17     private static final Logger LOGGER = Logger.getLogger(ScriptingDemo.class.getName());
18
19     public static void main(String[] args) {
20         ScriptEngineManager m = new ScriptEngineManager();
21         List<ScriptEngineFactory> factories = m.getEngineFactories();
22         factories.stream().map((factory) -> {
23             System.out.println(String.format("%s (%s %s)", factory.getLanguageName(),
24                                             factory.getEngineName(), factory.getImplementationName()));
25         });
26     }
27 }
```

The 'Output' tab at the bottom shows the build log:

```
run:
ECMAScript (Oracle Nashorn 1.8.0_192)
Hello ECMAScript
BUILD SUCCESSFUL (total time: 0 seconds)
```

Vielleicht doch Rhino?

- Aktuelle Version von der [Mozilla-Website](#) herunterladen
- Unter lib/ext ablegen (Extension Mechanism)

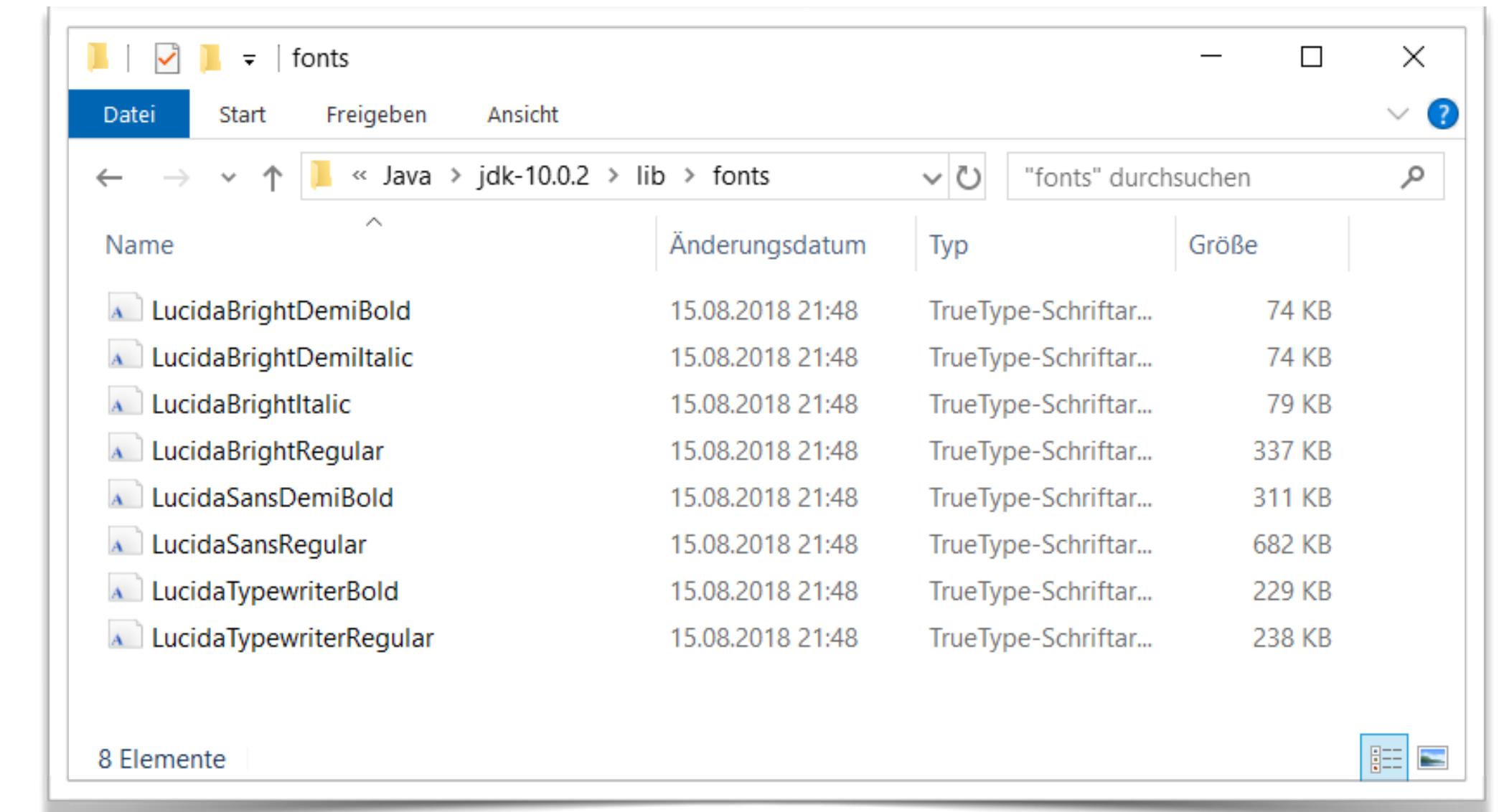
```
[TommisMacBookPro:dist thomas$ java -jar ScriptingDemo.jar  
<JAVA_HOME>/lib/ext exists, extensions mechanism no longer supported; Use -class  
path instead.  
.Error: Could not create the Java Virtual Machine.  
Error: A fatal exception has occurred. Program will exit.  
TommisMacBookPro:dist thomas$ ]  
  
wurde mit Java SE 9 entfernt
```

Tamensi movetur

- Aktuelle Rhino-Versionen enthalten die JSR223-Brücke nicht mehr
- Herunterladen über Maven-Suche: rhino-js-engine
- ```
java --module-path ScriptingDemo.jar:/Users/
thomas/Downloads/rhino-js-engine-1.7.7.1.jar:/
Users/thomas/Downloads/rhino1.7.9/lib/
rhino-1.7.9.jar --add-modules
rhino,rhino.js.engine --add-opens
rhino.js.engine/
com.sun.phobos.script.javascript=ALL-UNNAMED
com.thomaskuenneth.scriptingdemo.ScriptingDemo
```

# Textdarstellung

- Lucida Sans, Lucida Bright und Lucida Typewriter entfernt
- Lange in Java enthalten
  - Verfügbarkeit wird erwartet
  - Apps fallen auf Dialog zurück



# Ebenfalls Einfluss auf Darstellung

- Font Rasterization: Umstieg von T2K auf Freetype abgeschlossen
- Font Layout Engine: Umstieg von ICU nach HarfBuzz abgeschlossen
- Unterschied abhängig von Betriebssystem und alter Java-Version

JDK 11 Release Notes, Importan... +

https://www.oracle.com/technetwork/java/javase/11-relnote-issues-5012449.html#Removed

core-libs

→ **Removal of sun.misc.Unsafe.defineClass**

The `sun.misc.Unsafe.defineClass` class has been removed. Users should use the public replacement, `java.lang.invoke.MethodHandles.Lookup.defineClass`, added in Java SE 9. For more details see the Java documentation:

<https://docs.oracle.com/javase/9/docs/api/java/lang/invoke/MethodHandles.Lookup.html#defineClass-byte:A->

See [JDK-8193033](#)

core-libs/java.lang

→ **Removal of Thread.destroy() and Thread.stop(Throwable) Methods**

The methods `Thread.destroy()` and `Thread.stop(Throwable)` have been removed. They have been deprecated for several Java SE releases. The `Thread.destroy()` method has never been implemented, and the `Thread.stop(Throwable)` method has been non-functional since Java SE 8. No code should rely on the behavior of these two methods; however, any code that uses these methods will cause compilation errors. The mitigation is to remove references to these methods from the source code. Note that the no-arg method `Thread.stop()` is unaffected by this change.

See [JDK-8204243](#)

core-libs/java.nio

→ **Removal of sun.nio.ch.disableSystemWideOverlappingFileLockCheck Property**

The property `sun.nio.ch.disableSystemWideOverlappingFileLockCheck` has been removed. Consequently, compatibility with the older locking approach has also been removed.

JDK 6 introduced the system property `sun.nio.ch.disableSystemWideOverlappingFileLockCheck` to control file locking behavior. Specifically, the property was used to enable suppression of JVM-wide file locking and provide compatibility with JDK 1.4 and JDK 5. The old behavior was constrained to check for locks obtained only on the channel instance and not JVM-wide, which is what was actually specified.

See [JDK-8196535](#)

core-libs/java.util:i18n

# Lohnt Migration?

- Je nach Aktualität der App ggf. größere Umbauten
- Subtile Änderungen, zum Teil schwer zu finden  
Deshalb: mvn clean install
- Aber: nur weil der Code übersetzt wird, muss er noch lange nicht funktionieren  
Deshalb: Ausführlich testen

- Aktualisierung vermutlich günstiger als Neuentwicklung
- Behauptung: unabhängig von der Wahl der Technologie

# Und neue Apps?

- Wenn Java gewünscht...
  - Klassischer Rich Client: JavaFX
  - Browser-basiert: Java-JavaScript-Brücke oder Transpiler  
([JSweet](#), [Bck2Brwsr](#), ...)
  - Oder aktuelle Webtechniken

# Zusammenfassung

- Vorhandene Apps können ihre Frontend-Technologie weiterhin nutzen
- „Nicht im JDK“ bedeutet nicht „Ende der Straße“
- Einmalig Migration nach Java SE 11 nicht genug
  - Änderungen in Nicht-LTS-Versionen im Auge behalten
  - Begrenzter Support-Zeitraum auch für LTS-Versionen

# Vielen Dank!

<https://github.com/tkuenneth/java> on the client

thomas.kuenneth@mathema.de

@tkuenneth