31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

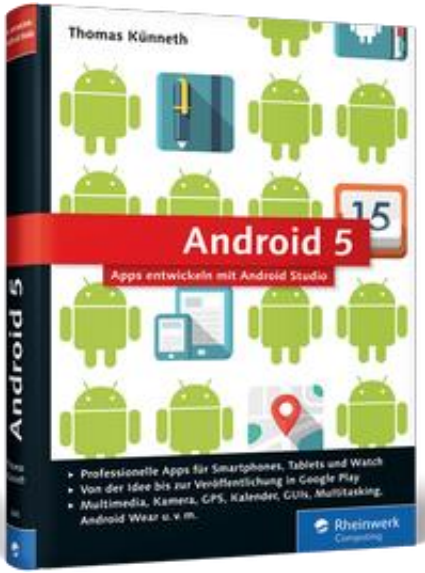
Nummer 5 lebt

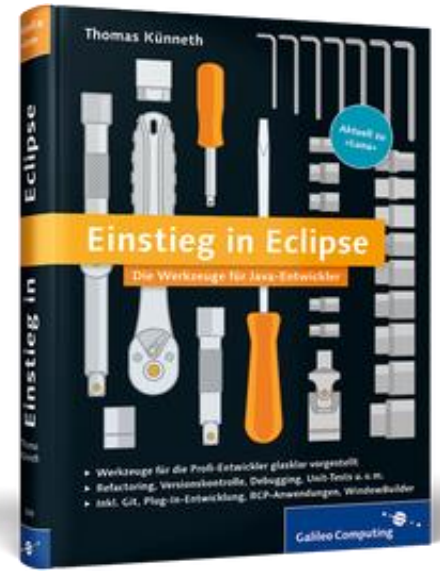
Aktuelle Java-Features unter Android nutzen

Thomas Künneth, M.A.

MATHEMA Software GmbH

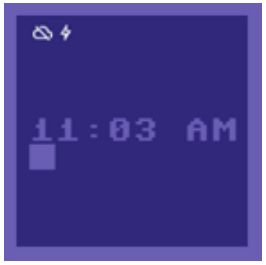
Über mich

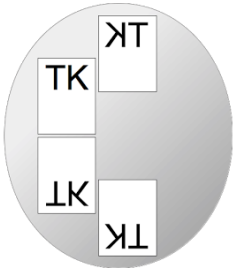

Thomas Künneth
Android 5
15
Apps entwickeln mit Android Studio
Android 5
Rheinwerk


Thomas Künneth
Einstieg in Eclipse
Die Werkzeuge für Java-Entwickler
Eclipse
Einstieg in
Galileo Computing


Thomas Künneth
Java
für Windows
Sold out
Galileo Computing


29
Aug


11:03 AM


TK
TK
TK
TK

Es ist allgemein bekannt, dass...

- ... die meisten Android-Apps in Java geschrieben werden
- Aber was bedeutet das eigentlich?
- In Java zu programmieren heißt...
  1. eine objektorientierte Programmiersprache nutzen
  2. Programme in einer virtuellen Maschine ausführen
  3. auf eine riesige (kaum mehr überblickbare) Klassenbibliothek zugreifen

Java-Evolution

- Sprache/Compiler, virtuelle Maschine und Klassenbibliothek wurden im Laufe der Jahre zuerst von Sun, dann Oracle konsequent weiterentwickelt
- Üblicherweise werden diese Bestandteile nicht einzeln wahrgenommen, sondern als Bündel (Java-Version)
- Zudem gibt es drei Java-Editionen (Mobile, Standard, Enterprise). Sie unterscheiden sich in Bezug auf die verwendete VM, Ablaufumgebung und Umfang der Klassenbibliothek.

Eine Nummer ziehen

Viele Java-Entwickler, die in Android einsteigen, fragen deshalb: „Welche Java-Version nutzt Android?“

System.getProperty()

```
public static void main(String[] args) {
    // Java Virtual Machine specification version/vendor/name
    outputProperty("java.vm.specification.version");
    outputProperty("java.vm.specification.vendor");
    outputProperty("java.vm.specification.name");
    // Java Virtual Machine implementation version/vendor/name
    outputProperty("java.vm.version");
    outputProperty("java.vm.vendor");
    outputProperty("java.vm.name");
    // Java Runtime Environment specification version/vendor/name
    outputProperty("java.specification.version");
    outputProperty("java.specification.vendor");
    outputProperty("java.specification.name");
    // Java Runtime Environment (implementation) version/vendor (no name)
    outputProperty("java.version");
    outputProperty("java.vendor");
    // Java class format version number
    outputProperty("java.class.version");
}

private static void outputProperty(String p) {
    System.out.println(p + " -> " + System.getProperty(p));
}
```

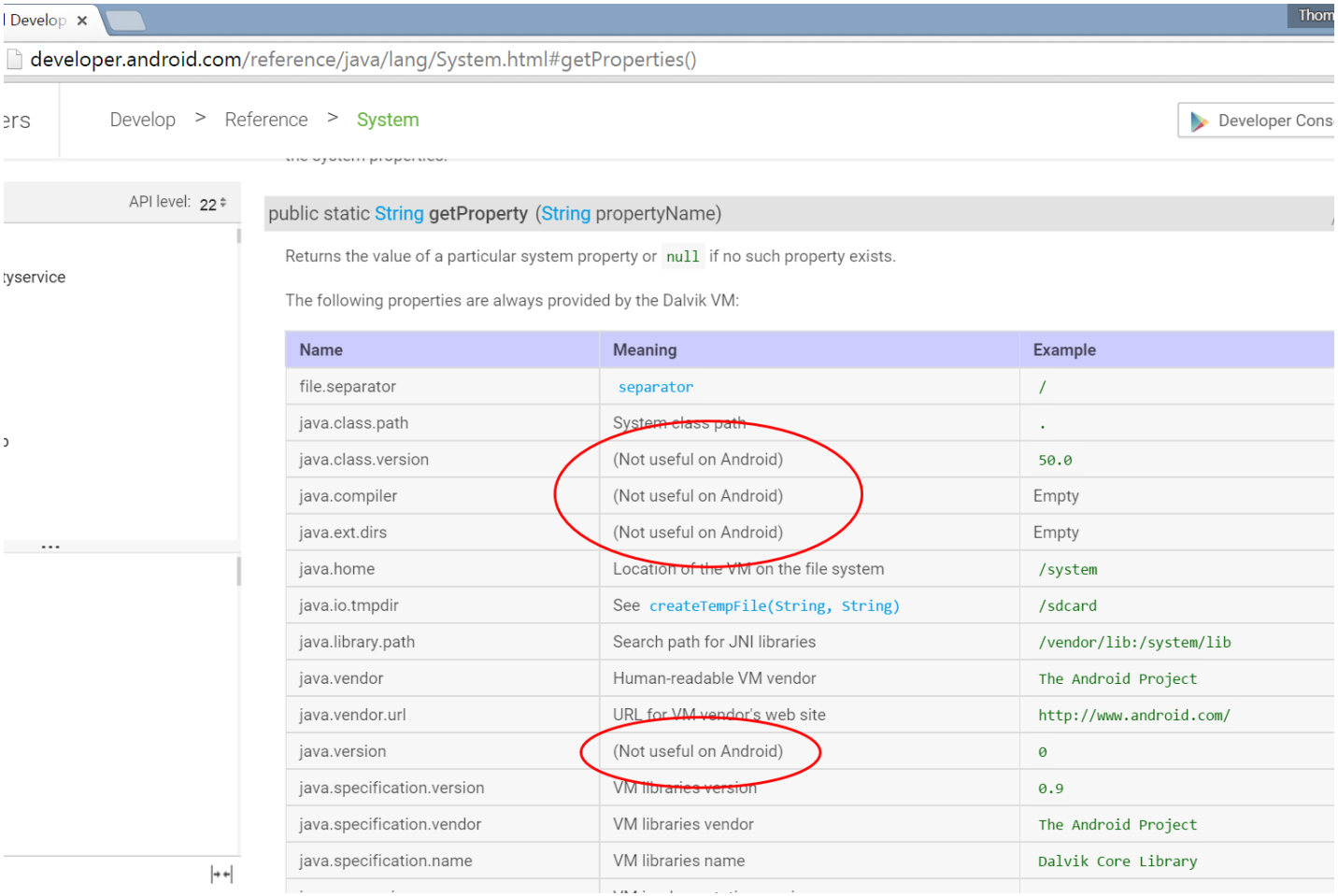
System Properties im Vergleich

com.thomaskuenneth.beyondharmony.BeyondHarmony > outputProperties >

Usages | Output - Beyond Harmony (run) ×

```
run:
java.vm.specification.version -> 1.8
java.vm.specification.vendor -> Oracle Corporation
java.vm.specification.name -> Java Virtual Machine Specification
java.vm.version -> 25.45-b02
java.vm.vendor -> Oracle Corporation
java.vm.name -> Java HotSpot(TM) 64-Bit Server VM
java.specification.version -> 1.8
java.specification.vendor -> Oracle Corporation
java.specification.name -> Java Platform API Specification
java.version -> 1.8.0_45
java.vendor -> Oracle Corporation
java.class.version -> 52.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

„Not useful“


Develop
developer.android.com/reference/java/lang/System.html#getProperties()
Develop > Reference > System
Developer Cons
API level: 22
public static String getProperty (String propertyName)
Returns the value of a particular system property or null if no such property exists.
The following properties are always provided by the Dalvik VM:
Name | Meaning | Example
file.separator | separator | /
java.class.path | System class path | .
java.class.version | (Not useful on Android) | 50.0
java.compiler | (Not useful on Android) | Empty
java.ext.dirs | (Not useful on Android) | Empty
java.home | Location of the VM on the file system | /system
java.io.tmpdir | See createTempFile(String, String) | /sdcard
java.library.path | Search path for JNI libraries | /vendor/lib:/system/lib
java.vendor | Human-readable VM vendor | The Android Project
java.vendor.url | URL for VM vendor's web site | http://www.android.com/
java.version | (Not useful on Android) | 0
java.specification.version | VM libraries version | 0.9
java.specification.vendor | VM libraries vendor | The Android Project
java.specification.name | VM libraries name | Dalvik Core Library

Welche Sprachfeatures sind vorhanden?

- Die Frage sollte deshalb nicht lauten „Welche Java-Version verwendet Android?“
- Sie lässt sich nicht mit einem Wort, oder einer Zahl beantworten.

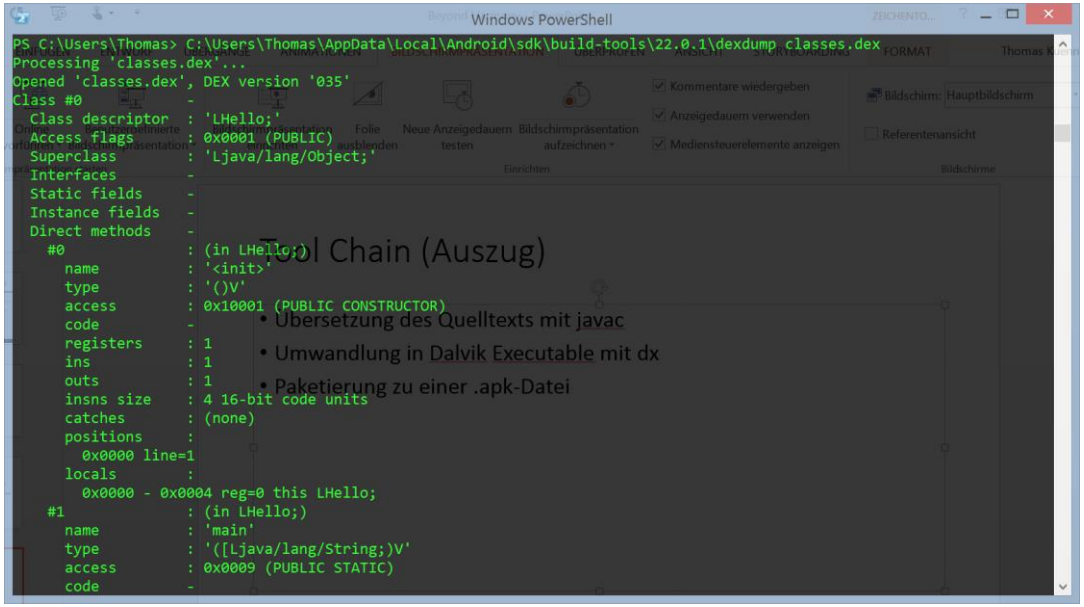
- Viel wichtiger ist:
  - Welche Sprachfeatures kann ich nutzen (Generics, Lambda-Ausdrücke, …)?
  - Welche besonderen Eigenschaften hat die verwendete virtuelle Maschine (Verhalten des Garbage Collectors, Instrumentation, ...)?
  - Welche Klassen und Pakete sind vorhanden?

Keine klassische JVM

- Google hat Java nicht von Sun oder Oracle lizensiert
- Android verwendet keine virtuelle Maschine, die den vom Java Compiler erzeugten Bytecode direkt ausführen kann
- Android enthielt bis Version 4.4 eine Eigenentwicklung namens Dalvik
  - Keine Stapel-, sondern Registermaschine
  - Eigener, unabhängiger Befehlssatz
  - Für Geräte mit begrenzten Ressourcenumfang konzipiert
- Ab Android 5 ausschl. Erzeugung „echter“ Maschinensprache aus Dalvik-Code durch Ahead of time-Compiler

Der Umwandlungsprozess

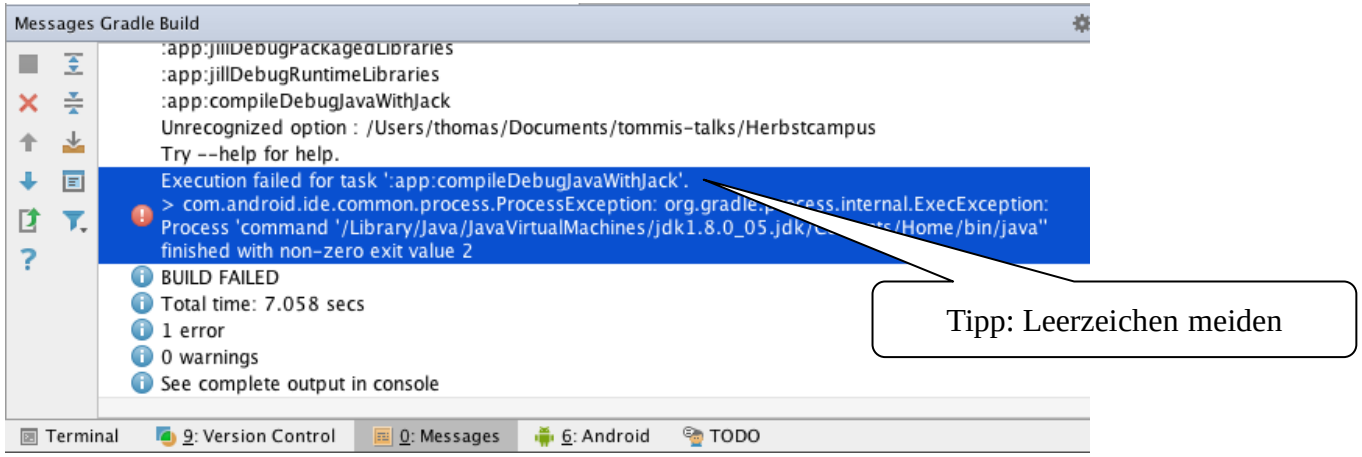
- Übersetzung des Quelltexts mit javac
- Umwandlung in Dalvik Executable mit dx
- Alle Klassen landen in .dex-Dateien
- Paketierung zu einer .apk-Datei



Anzeige von .dex-Dateien mit dexdump (vergl. mit javap)

Jack and Jill

- im Dezember 2014 hat Google eine neue, experimentelle toolchain vorgestellt
- Ziel: schnellere Build-Zeiten, weniger Abhängigkeit von anderen Tools
- Kern der neuen toolchain:
  - Jack (Java Android Compiler Kit)
  - Jill (Jack Intermediate Library Linker)



Und so funktioniert‘s

- Jill wandelt referenzierte Bibliotheken in neue Jack-Bibliotheksdateien (.jack) um
- können schnell mit anderen .jack-Files gemischt werden
- Android Gradle-Plug-in und Jack sammeln alle .jack-Bibliotheksdateien und den Quellcode der App zusammen und übersetzen sie in .dex-Dateien.
- Daraus wird wie gewohnt eine .apk-Datei erzeugt

```
android {
  ...
  // buildToolsRevision >= '21.1.1'
  defaultConfig {
    // Stand 30.08.15:
    // nicht automatisch aktiv
    useJack = true
  }
  ...
}
```

Abhängigkeiten durch Konvertierung

- Neue Java-Sprachfeatures können Auswirkungen auf die von javac generierten .class-Dateien haben
- Beispiel: Bei der Umsetzung von Lanbdas greift javac unter anderem auf das mit Java 7 eingeführte invokedynamic zurück
- Android's „altes" Konvertierungstool muss also mindestens invokedynamic kennen, um Lambdas in Dalvik-Bytecode umsetzen zu können
- der „neue" Compiler Jack muss ebenfalls neue Sprachfeatures kennen

Lambdas

- Wurde mit Java 8 eingeführt
- Die Android Studio-Funktion „Code Folding“ wandelt bestimmte anonyme Klassen für die **Anzeige** im Editor um

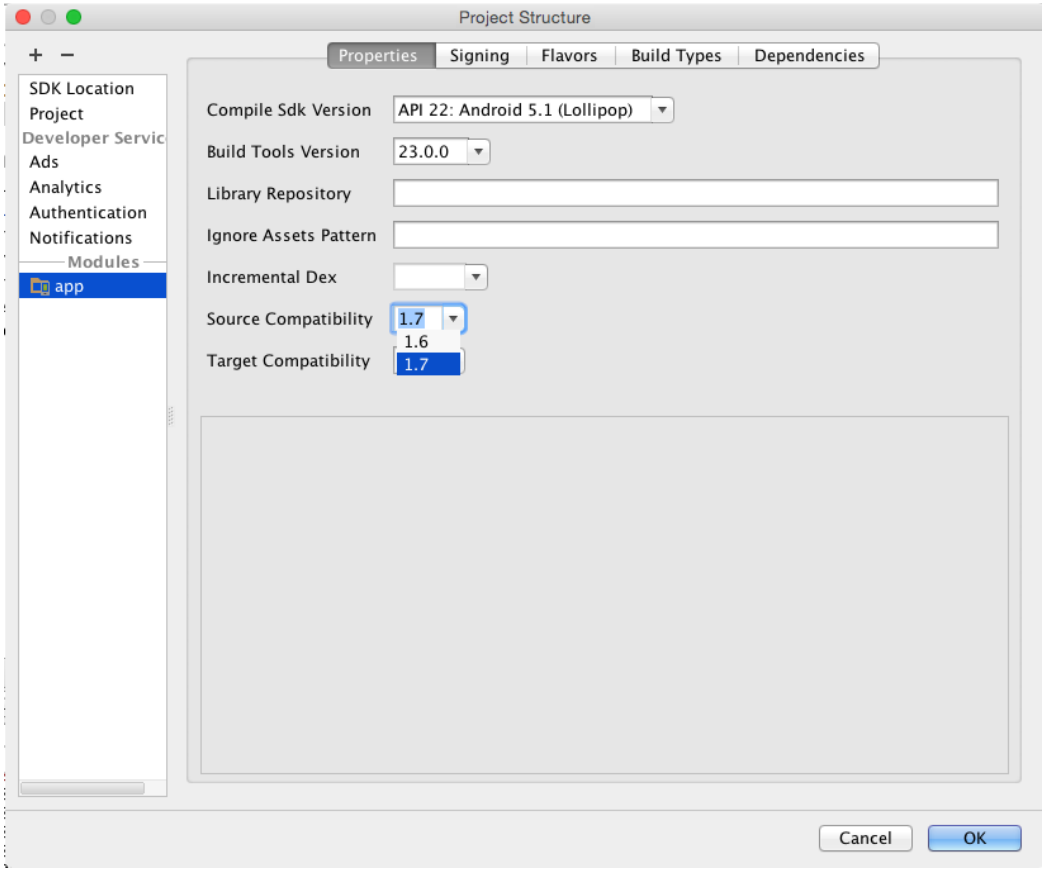
```
tv.setOnLongClickListener((v) → {
        Toast.makeText(LanguageFeatureDemo.this,
                getString(R.string.hello), Toast.LENGTH_LONG)
                .show();
        return false;
});
```

- Aber kann man auch Java-Lambdas in Apps verwenden?

- Stand 30.08.2015: nein (Exceptions in der Toolkette); auch Jack bietet derzeit nur Support für Java 7

Kompatibilitätseinstellung anpassen

- Um neue Sprachfeatures zu nutzen, muss die Kompatibilitätsstufe angepasst werden
- Stand 30.08.2015 ist 1.8 noch nicht vorhanden, muss deshalb direkt in build.gradle editiert werden
- Frage: Warum?



Retrolambda für Java 6 und 7

- Retrolambda von Esko Luontola kann Lambdas automatisch zurückportieren
- Gradle-Plugin von Evan Tatarka: nur wenige Anpassungen in zwei build.gradle-Dateien nötig

```
public class RetrolambdaDemo extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_retrolambda_demo);
        TextView tv = (TextView) findViewById(R.id.textview);

        StringBuilder sb = new StringBuilder();
        Integer[] numbers = {3, 1, 9, 5, 7};

        Arrays.sort(numbers, (Integer i1, Integer i2) -> {
            return i2 - i1;
        });

        for (Integer i : numbers) {
            if (sb.length() > 0) {
                sb.append(", ");
            }
            sb.append(i);
        }
        tv.append(sb.toString());
    }
}
```

Retrolambda unter Android

Demo und Code

(Android Studio: RetrolambdaDemo)

Strings in switch-Anweisungen

- Strings in switch-Anweisungen wurden mit Java 7 eingeführt
- Kann den Code „schöner“ machen
- Hat zu vielen Diskussionen bzgl. Sinn und Unsinn geführt

```
3   public class StringSwitch {
4
5       public static void main(String [] args) {
6           String s = "two";
7           int e = stringSwitch(s);
8           System.out.println(s + " ist " + e);
9       }
10
11      public static int stringSwitch(String string) {
12          switch (string) {
13              case "one":
14                  return 1;
15              case "two":
16                  return 2;
17              case "three":
18                  return 3;
19              default:
```

try-with-resources

- Automatisches Ressourcenmanagement im Kontext try-with-resources; wurde mit Java 7 eingeführt
- Macht den Code schlanker
- Entwickler können das Schließen eines Stroms nicht mehr vergessen

```
public class TryWithResources {

    private static final String CLASSNAME = TryWithResources.class.getName();
    private static final Logger LOGGER = Logger.getLogger(CLASSNAME);

    public static void main(String[] args) {
        File file = new File(System.getProperty("user.home"), "file.txt");
        String contents = "Hello MATHEMA";
        try (
                FileWriter fw = new FileWriter(file) {
                    @Override
                    public void close() throws IOException {
                        super.close();
                        LOGGER.log(Level.INFO, "close() called");
                    }
                };) {
            fw.write(contents);
        } catch (IOException e) {
            LOGGER.throwing(CLASSNAME, "oh no...", e);
        } finally {
            long len = file.length();
            file.delete();
            LOGGER.log(Level.INFO, "Länge: {0}", new Object[]{len});
        }
    }
}
```

nur für Demozwecke nötig

Neue Sprachfeatures unter Android

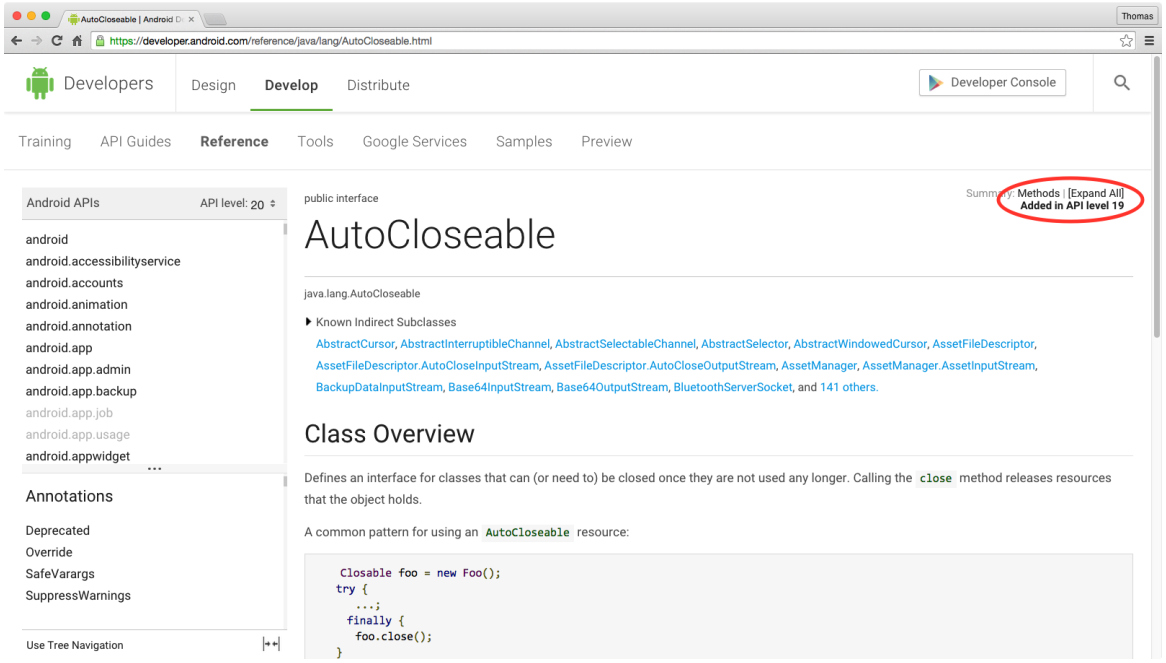
Demo und Code

(Android Studio: LanguageFeatureDemo)

Funktioniert prima unter Android, wenn…

- …Source Compatibility und Target Compatibility auf 1.7…
- … **und** minSdkVersion auf mindestens 19 gesetzt werden.

- AutoCloseable wurde erst mit API-Level 19 in Android übernommen



Abhängigkeiten

- Manche Java-Sprachfeatures haben also (möglicherweise nicht offensichtliche) Abhängigkeiten zur Klassenbibliothek

- Daraus ergibt sich unweigerlich die Frage:

  „Welche Klassenbibliothek nutzt Android?“

Apache Harmony

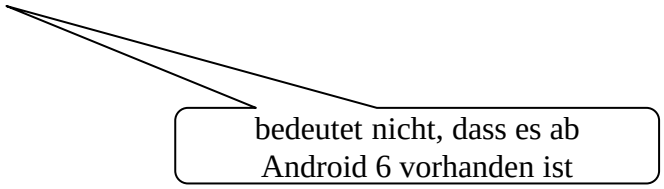
- Harmony war eine freie Java SE-Implementierung der Apache Software Foundation, einschl. vm, Compiler/Tools und Klassenbibliothek
- Zwei Releasestränge: J2SE 5.0 und Java SE 6
- Ziel war, alle Entwickler eines open source-Javas unter einem Dach zu vereinen. Trat damit in Konkurrenz zu GNU Classpath.
- Seit 25.10.2006 Apache Toplevel Projekt
- Von umfangreichen Code-Spenden großer Firmen (z. B. IBM und Intel) abhängig
- Von Beginn an Probleme, eine (aus Apache-Sicht geeignete) Lizenz für das Technology Compatibility Kit (TCK) zu erhalten
- Mit OpenJDK fiel faktisch die Notwendigkeit für das Projekt weg; es wurde am 16.11.2011 eingestellt

Was bedeutet das?

- Viele Teile der Android-Klassenbibliothek stammen aus Harmony
- Alle Android-spezifischen Klassen und Pakete stammen direkt von Google. Deren Verfügbarkeit ist durch den API Level definiert.
- Zahlreiche Klassen, Methoden und Pakete hat Google nachträglich in den ehemaligen Harmony-Code eingefügt
- Allerdings übernimmt Google bei Weitem nicht alle Aktualisierungen der Java-Standardklassenbibliothek
- Was vorhanden ist, kann also nicht von **einer Versionsnummer** abgeleitet werden, sondern muss durch Ausprobieren oder Stöbern in der Doku erschlossen werden

Ein paar willkürliche Beispiele

- ClassLoader.registerAsParallelCapable() (Java 7) bis einschl. API Level 22 nicht vorhanden
- Modifier.classModifiers() (Java 7) erst ab API Level 19 vorhanden
- javax.script.ScriptEngine (Java 6) bis einschl. API Level 22 nicht vorhanden
- java.time.LocalDateTime (Java 8) bis einschl. API Level 22 nicht vorhanden



Java Compiler API (JSR 199)

Demo und Code

(NetBeans: CompilerAPIDemo)

unter Android nicht vorhanden

Fehlende Features nachrüsten

- Viele vermisste Features sind aus Java Specification Requests hervorgegangen
- Für viele JSRs gibt es Referenzimplementierungen
- Deshalb liegt die Frage nahe:

  „Können diese unter Android eingesetzt werden?“

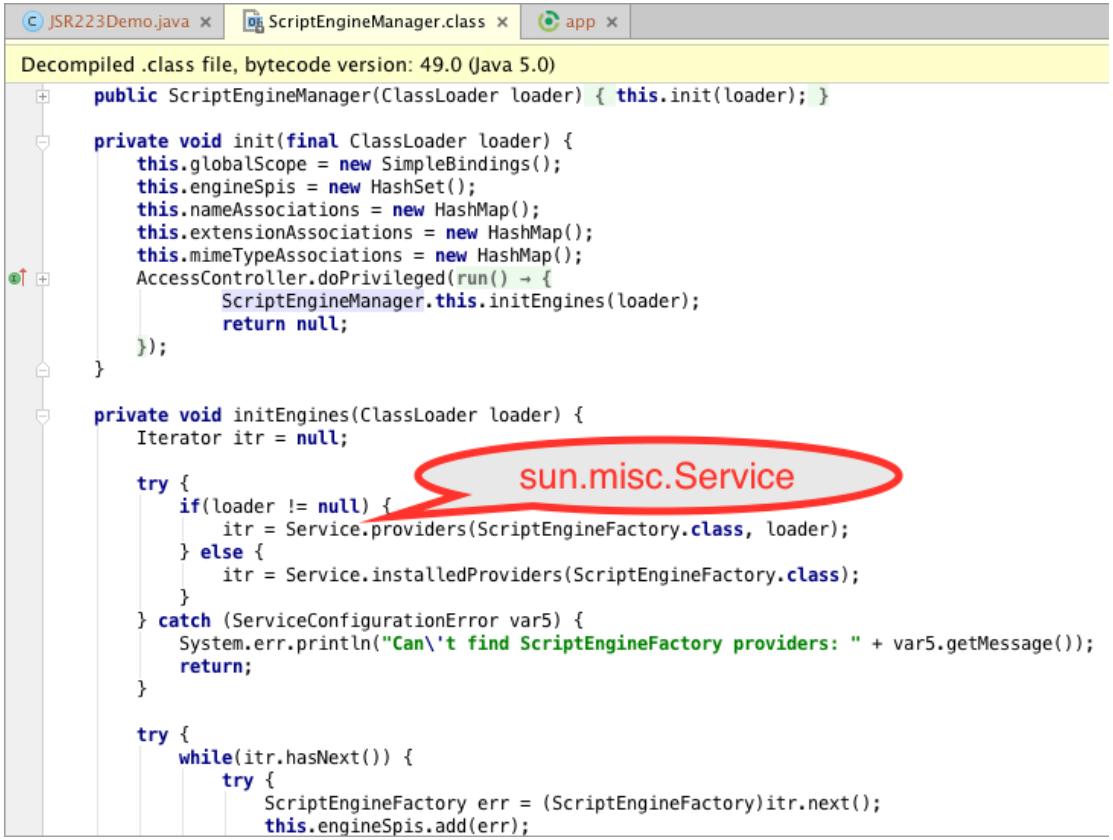
Integration von Scriptsprachen (JSR 223)

Demo und Code

(Android Studio: JSR223_RI)

sun.misc.Service

- JSR 223 RI nicht „out of the box“ nutzbar, weil eine benötigte Klasse (sun.misc.Service) unter Android nicht vorhanden ist
- Stattdessen gibt es ab Java 6 die Klasse java.util.ServiceLoader (auch unter Android vorhanden)



Minimale Eigenimplementierung

```
package sun.misc;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.ServiceLoader;

/**
 * This class implements just one specific method, which the reference implementation of JSR 223
 * (at least) needs.
 * If you are interested, take a look at initEngines() in javax.script.ScriptEngineManager.
 *
 * @author Thomas Kuenneth
 */
public class Service {

    private static final Map<Class<?>, ServiceLoader<Object>> MAP = new HashMap<Class<?>, ServiceLoader<Object>>();

    public static Iterator providers(Class service, ClassLoader loader) {
        if (!MAP.containsKey(service)) {
            MAP.put(service, ServiceLoader.load(service, loader));
        }
        ServiceLoader sl = MAP.get(service);
        return sl.iterator();
    }
}
```

Integration von Scriptsprachen (JSR 223)

Demo

(Android Studio: JSR223_RI)

Zweiter Versuch

Warum es dennoch nicht geht

- UnsupportedOperationException("can't load this type of class file“) nach Aufruf von loader.defineClass(className, classBytes) in private Class defineClass(Object bytecode, Object staticSecurityDomain) (org.mozilla.javascript.optimizer.Codegen)
- Android mag den „on the fly“ erzeugten Bytecode nicht

Oder vielleicht doch?

- Anstelle von Rhino könnte ein anderer JavaScript-Interpreter verwendet werden
- In Android‘s WebView-Komponente ist JavaScript nutzbar
- Das open source-Projekt jjsb4a bildet eine Brücke zwischen JSR 223 und WebView

Java-JavaScript-Bridge for Android

Demo und Code

(Android Studio: JSR223_RI)

Date and Time API (JSR 310)

- Wurde mit Java 8 Bestandteil der Plattform
- Referenzimplementierung (Project ThreeTen) steht auch als Backport für Java SE 6 und 7 zur Verfügung
- Backport ist aber laut Projekt-Homepage keine offizielle Implementierung von JSR 310 (u. a. abweichende Paketnamen)
- Kann sehr einfach in Android-Projekte integriert werden

```
repositories {
  mavenCentral()
}

dependencies {
  compile fileTree(include: ['*.jar'],
                   dir: 'libs')
  compile 'org.threeten:threetenbp:1.3'
}
```

ThreeTen-Backport

Demo und Code

(Android Studio: ThreeTen-Demo)

Zusammenfassung...

- Es gibt viele weitere Aspekte der Java-Klassenbibliothek, die nicht „out of the box“ verfügbar sind
- Handelt es sich um Java Specification Requests, lohnt ein Blick auf die Referenzimplementierung, insb. wenn diese für Java SE 5, 6 oder 7 entwickelt wurde
- Auch für Teile von JSRs kann es Backports geben. Beispiel: Java Streams API
- Aber: nicht jede Implementierung läuft unter Android. Beispiel: JAX-WS-RI (Blogpost vom 14. April 2015)

... und Ausblick

- Neue Java-Features erreichen Android immer mit Verspätung
- Bislang war Google bemüht, Sprachfeatures verfügbar zu machen
- Selbst neue Klassen der Standardbibliothek werden gelegentlich übernommen
- Aber: kein Plan erkennbar; die Übernahmen wirken willkürlich
- Und: die Schere öffnet sich weiter. Android entfernt sich vom „Standard Java“
- Ob Google wie Apple eine neue Sprache einführt, ist derzeit ungewiss
  - Go vermutlich zu unbekannt
  - Dart: Projekt „Dart on Android“ (Sky)

Links

- **Jack and Jill** http://android-developers.blogspot.de/2014/12/hello-world-meet-our-new-experimental.html
- **Retrolambda** https://github.com/orfjackal/retrolambda
- **Gradle Plug-in** https://github.com/evant/gradle-retrolambda
- **JSR 223** https://jcp.org/aboutJava/communityprocess/final/jsr223/index.html
- **Java-JavaScript-Bridge-for-Android** https://github.com/tkuenneth/jjsb4a
- **JSR 310** https://jcp.org/en/jsr/detail?id=310
- **Project ThreeTen** http://www.threeten.org/
- **Backport** http://www.threeten.org/threetenbp/
- **JSR 199** https://jcp.org/en/jsr/detail?id=199
- **Seifenkisten auf Abwegen** http://kuennetht.blogspot.de/2015/04/seifenkisten-auf-abwegen.html

Fragen, Anregungen, Diskussion

Vielen Dank!

Thomas Künneth, M.A.

MATHEMA Software GmbH

thomas.kuenneth@mathema.de

@tkuenneth

http://kuennetht.blogspot.de/