Von Geysiren und Kaffeebohnen

Eine kleine Tour durch die Android-Laufzeitumgebung

Jörg Pleumann
Noser Engineering AG

Android Experience Day 2008

Agenda

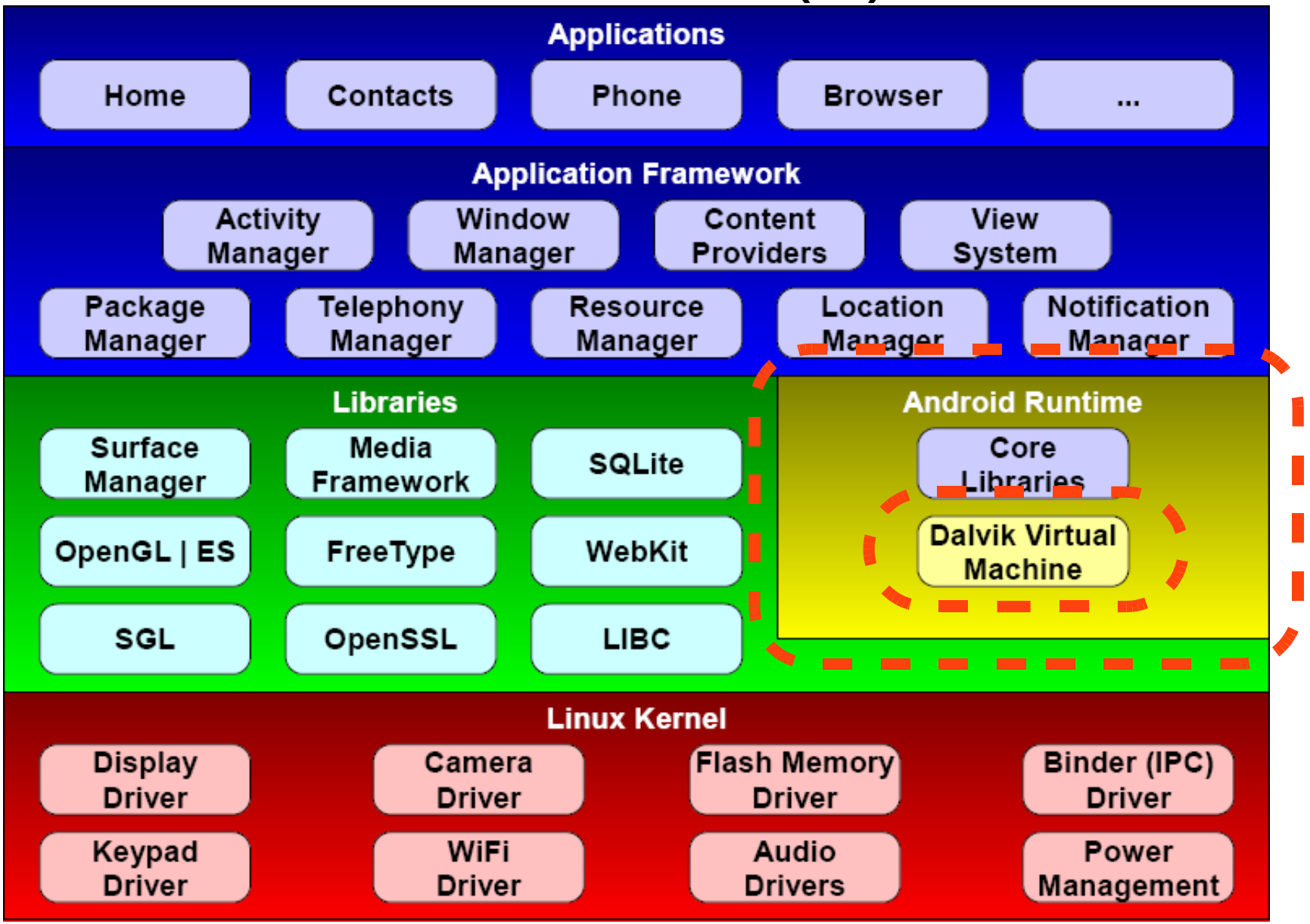
- Überblick
- Virtuelle Maschine
- Core Libraries
- Zusammenfassung

Überblick (I)

- Was ist Android?
  - Kompletter Software Stack für mobile Geräte
- Erste Näherung
  - Linux-Kernel
  - Programmierung in Java
  - Virtuelle Maschine (die keine JVM ist)
  - Neues Framework für Applikationen

Wie passt das alles zusammen?

Überblick (II)



Dalvik VM (I)

noser

- Bytecode Interpreter für mobile Systeme
  - Langsame CPU (250-500 MHz)
  - Wenig RAM (64 MB)
  - Keine Auslagerungsmöglichkeit
  - Batteriebetrieb
- Eine Instanz pro laufender Anwendung
- Effizienz sehr wichtig
  - CPU
  - Speicher

10 Jahre alter PC

Dalvik VM (II)

- Üblich: Stack-Architektur
  - Operanden und Ergebnisse auf einem Stack
  - Zusätzlich lokale Variablen
- Interpreter (stark vereinfacht)

```
while (true) {
    char c = fetchAndDecode();
    switch (c) {
        case '#': doPush();  break;
        case '+': doAdd();   break;
        case '-': doSub();   break;
        ...
    }
}
```

- Dispatch bedeutet Overhead

Dalvik VM (III)

noser

- Dalvik: Register-Architektur
  - Alle temporären Werte in Registern
  - Konsequentere Architektur
- Effizienterer Interpreter
  - Höhere semantische Dichte der Instruktionen
  - Weniger Instruktionen benötigt
  - Spezielle Instruktionen für Problemfälle
- Kein Just-In-Time Compiler
  - Nicht so relevant

Befehlssatz (I)

- Beispiel

Der Klassiker...

```
System.out.println("Hallo Welt!");
```

JVM Bytecode

```
getstatic java.lang.System.out Ljava/io/PrintStream;
ldc "Hallo Welt!"
invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
return
```

DVM Bytecode

```
sget-object v0, java.lang.System.out Ljava/io/PrintStream;
const-string v1, "Hallo Welt!"
invoke-virtual {v0, v1}, java/io/PrintStream/println(L...;)V
return-void
```

Befehlssatz (II)

- Höhere semantische Dichte der Instruktionen

Beispiel: Einfache Schleife über ein Array

```
public static long sumArray(int[] arr) {
    long sum = 0;

    for (int i : arr) {
        sum += i;
    }

    return sum;
}
```

Befehlssatz (III)

- Höhere semantische Dichte der Instruktionen

JVM Bytecode

```
000b: iload 05
000d: iload 04
000f: if_icmpge 0024
0012: aload_3
0013: iload 05
0015: iaload
0016: istore 06
0018: lload_1
0019: iload 06
001b: i2l
001c: ladd
001d: lstore_1
001e: iinc 05, #+01
0021: goto 000b
```

30% Instruktionen weniger (Durchschnitt)

DVM Bytecode

```
0007: if-ge v0, v2, 0010
0009: aget v1, v8, v0
000b: int-to-long v5, v1
000c: add-long/2addr v3, v5
000d: add-int/lit8 v0, v0, #int 1
000f: goto 0007
```

Befehlssatz (IV)

- Spezielle Instruktionen für Problemfälle

Beispiel: Initialisierung eines Arrays

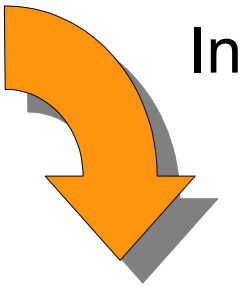
```
public class Demo {
    private static final char[] DATA = {
        'N','o','s','e','r',
        'k','n','o','w','s',
        'A','n','d','r','o','i','d'
    };
}
```

Befehlssatz (V)

- Spezielle Instruktionen für Problemfälle

JVM Bytecode

```
  0: bipush  17
  2: newarray char
  4: dup
  5: iconst_0
  6: bipush  78
  8: castore
       ...
 94: dup
 95: bipush  16
 97: bipush  100
 99: castore
100: putstatic DATA
103: return
```



Initialisierung des Arrays aus einer Tabelle

DVM Bytecode

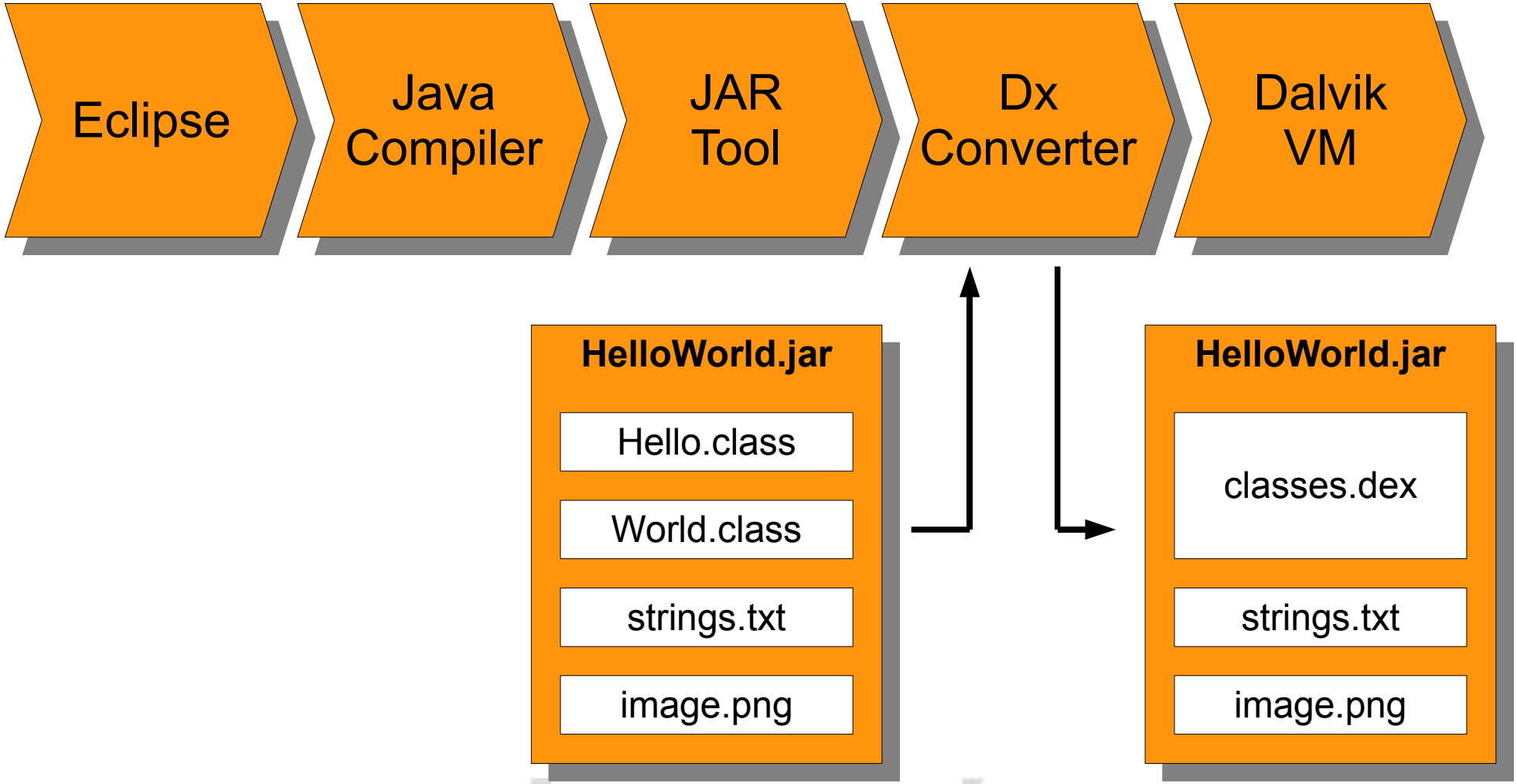
```
0000: const/16 v0, #int 17
0002: new-array v0, v0, [C
0004: fill-array-data v0, 0a
0007: sput-object v0, DATA:[C
0009: return-void
000a: array-data (21 units)
      'N', 'o', 's', 'e', 'r' ...
```

Binärformat (I)

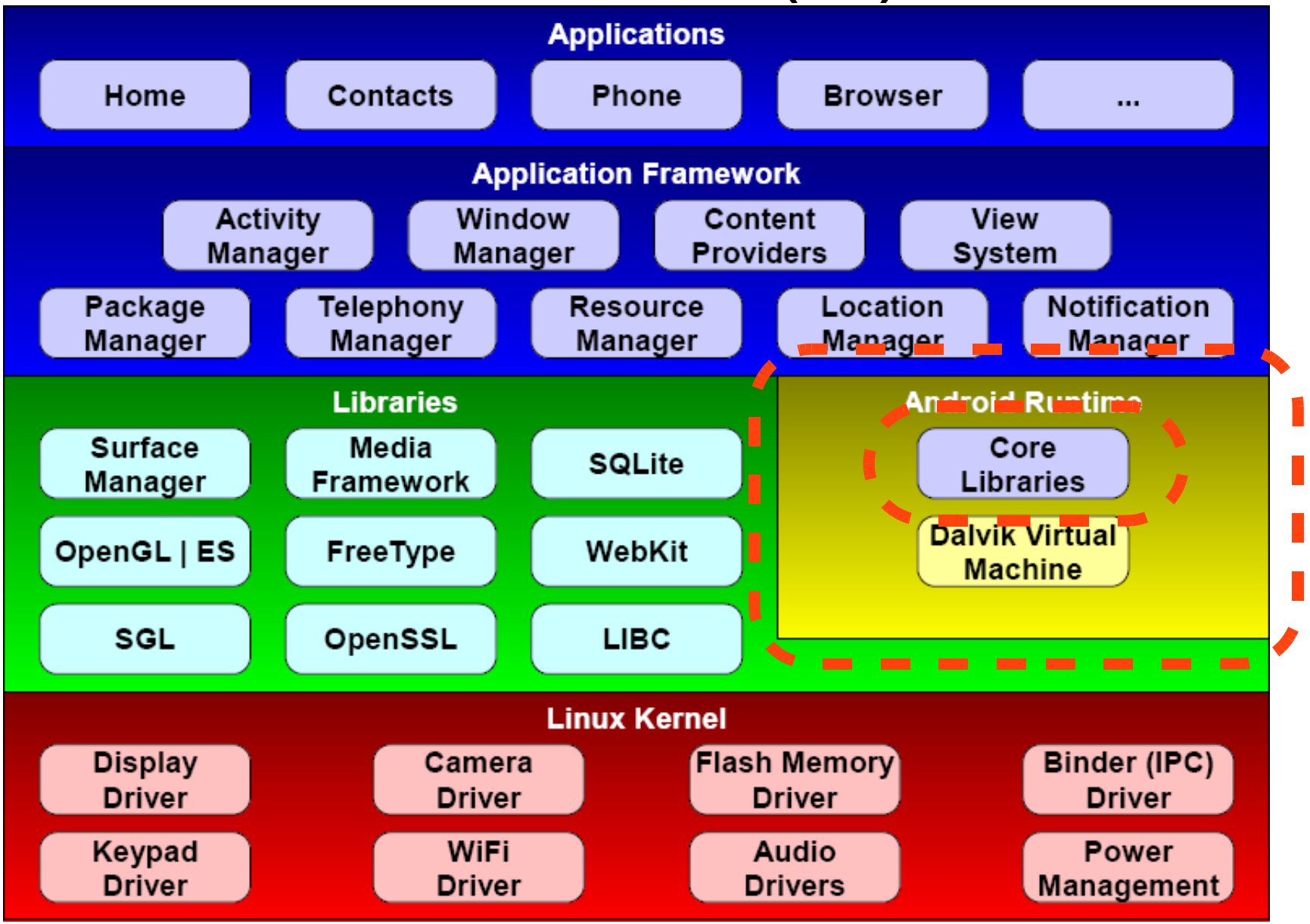
- Dalvik Executable Format (DEX)
- Größenreduktion
  - Weniger Instruktionen
  - Mehrere Klassen in einer Datei
  - Gemeinsame Konstanten-Pools
- Keine Kompression
  - Trotzdem im Normalfall kleiner als JAR
  - Erlaubt Nutzung von `mmap()`

Binärformat (II)

- Entwicklungsprozess



Überblick (III)



Core Libraries (I)

- Eng mit der VM verknüpfte Bibliotheken
  - Auf jedem Gerät vorhanden
  - Basis für das Android Framework
- Zerfallen in drei Teile
  - Dalvik-spezifische Pakete
    - System-Info, Debugger, ...
  - Übliche Java-Pakete
    - Detaillierte Betrachtung
  - Zusätzliche Pakete
    - Apache HttpClient 4.0

Core Libraries (II)

noser

Vollständig

java.io
java.lang
java.lang.annotation
java.lang.ref
java.lang.reflect
java.math
java.net
java.nio
java.nio.channels
java.nio.channels.spi
java.nio.charset
java.nio.charset.spi
java.security
java.security.acl
java.security.cert
java.security.interfaces
java.security.spec
java.sql

Vollständig

java.text
java.util
java.util.concurrent
java.util.concurrent.atomic
java.util.concurrent.locks
java.util.jar
java.util.logging
java.util.prefs
java.util.regex
java.util.zip

javax.crypto
javax.crypto.interfaces
javax.crypto.spec
javax.net
javax.net.ssl
javax.security.cert
javax.sql

Vollständig

org.xml.sax
org.xml.sax.ext
org.xml.sax.helpers

Ältere Version

javax.xml
javax.xml.parsers

org.w3c.dom

Unvollständig

javax.security.auth
javax.security.auth.callback
javax.security.auth.login
javax.security.auth.x500

Core Libraries (III)

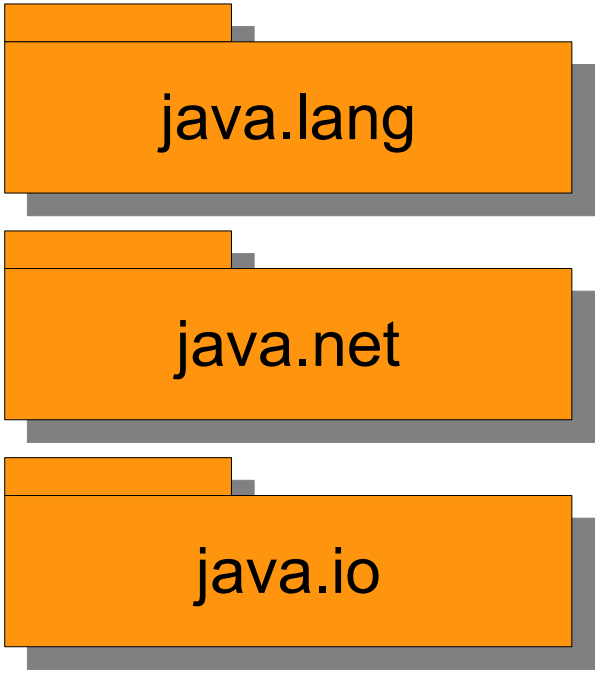
noser

- Beobachtung
  - Offenbar keine Micro Edition
  - Entspricht eher einem Desktop Java
  - GUI unberücksichtigt (kein AWT / Swing)
- Geht es präziser?
  - Android folgt keinem JSR-Modell
  - „Hinreichend kompatibel“ zu JDK 1.5

Wo liegen dann die Unterschiede?

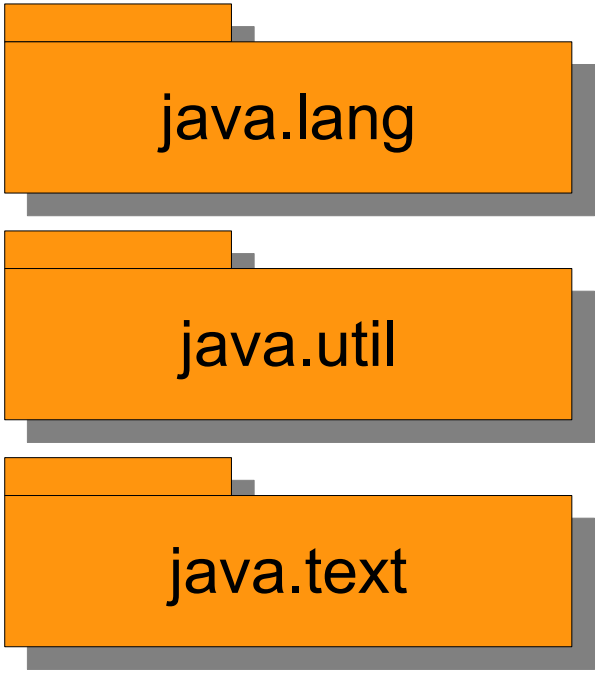
Zentrale Pakete

- ClassLoader basiert auf DEX
  - Kein defineClass(byte[])
  - Keine Instrumentierung
- Threads
  - Einige @deprecated-Methoden nicht implementiert
- Android / Linux-Security
  - Eine User-ID pro Applikation
- Kein Multicast Socket



Internationalisierung

- Basiert auf ICU 3.8.1
- Minimal andere Daten als JDK
- Unterschiede im Verhalten von
  - java.lang.Character
  - java.util.Locale
  - java.util.Formatter
- Nur begrenzte Menge von Ländern / Sprachen in Android enthalten

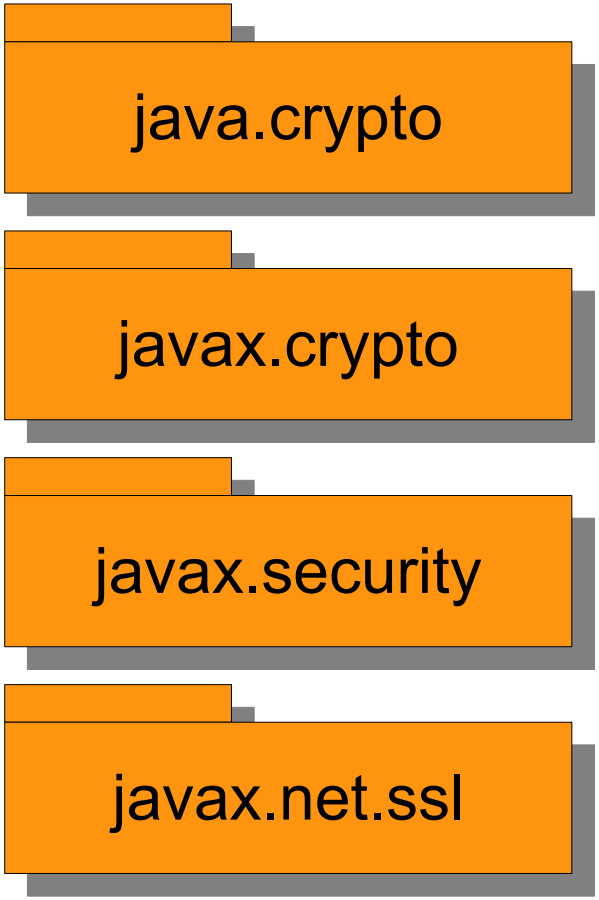


Reguläre Ausdrücke

- Basiert auf ICU 3.8.1
- Effiziente Implementierung
- Syntax / Semantik
  - Minimale Unterschiede zum JDK
  - Kollisionen unwahrscheinlich
  - Vorsicht bei existierendem Code

Sicherheit

- Hybride Implementierung
  - Bouncy Castle als JCE Provider
  - OpenSSL für zeitkritische Fälle
- Unterstützte Algorithmen
  - Menge entspricht nicht JDK
  - Alle wesentlichen vorhanden
  - Andere über SPI
- Bouncy Castle (.bks) Keystore
  - Kein eigenes keytool



Datenbank

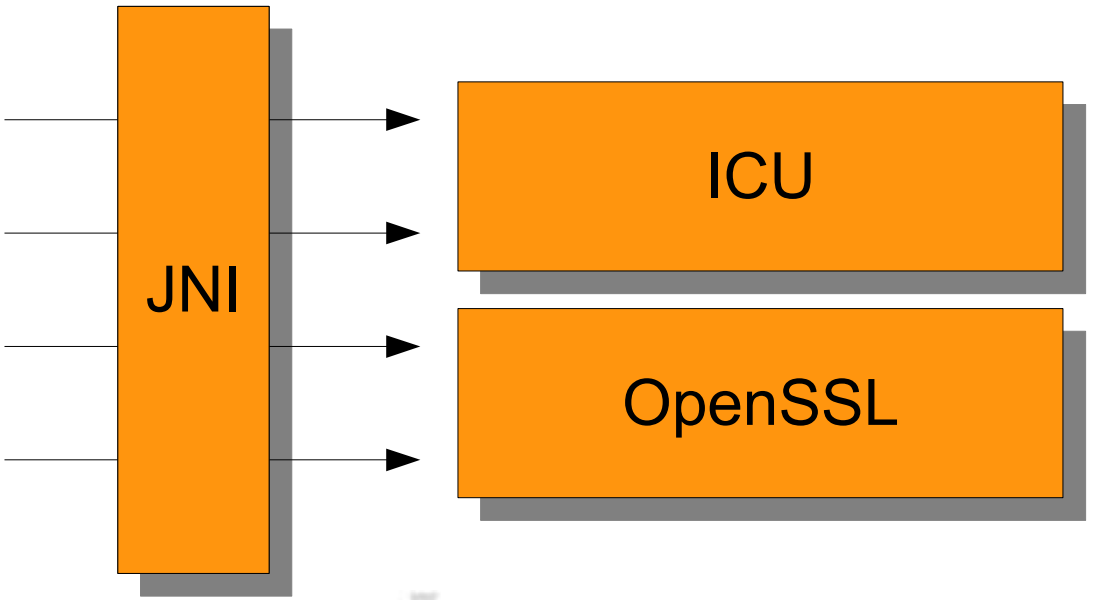
- JDBC 2.0
- SQLite-Treiber eingeschränkt
  - Nicht alle Datentypen
  - Diverse Methoden von ResultSet werfen Exceptions
  - Andere Treiber über SPI
- Alternative
  - SQLite-Schnittstelle in android.database.sqlite
  - Bessere Integration mit Aktivitäten

XML

- DOM Level 2 Core
  - Kein XPath, XSL, ...
  - Etwa Stand von 2001
  - Angemessen für mobile Anwendungen
- SAX Version 2
  - SAXParser / DocumentBuilder nicht validierend
  - Basiert auf KXML2 (bald Expat)
  - Andere Implementierungen über SPI

Implementierung

- Beigesteuert von Noser Engineering
  - 1900 API-Klassen, 3200 total (ohne HttpClient)
  - Teilweise portiert von Apache Harmony
  - Teilweise „from scratch“ entwickelt
- Optimierung
  - java.util.regex
  - java.text
  - java.security
  - java.math



Zusammenfassung (I)

- Dalvik VM
  - Effizienter Bytecode-Interpreter
  - Register-Architektur
  - Führt transformierten Java-Bytecode aus
- Core Libraries
  - Reichhaltige Java-Bibliothek
  - Pakete java.* und javax.*
  - Orientiert sich am Desktop

Zusammenfassung (II)