
mitp
Elisabeth
Jung
Android 4
Übungsbuch für die App-Entwicklung
Aufgaben mit vollständigen Lösungen

Architektur und Installation von Android-Applikationen

1.1 Native Apps versus Web-Apps

Die mit dem Android-Betriebssystem entwickelten Applikationen (auch als Apps benannt) gehören zur Gruppe der sogenannten »nativen Applikationen«, weil sie auf bestimmte Geräte und Software zugeschnitten sind, mit anderen Worten, sie sind plattformabhängig. Ein weiteres bekanntes Betriebssystem, auf dem Apps entwickelt werden, ist Apple iOS, das die Entwicklung von Apps überhaupt geprägt hat.

Webapplikationen sind Programme, die auf einem Webserver ablaufen und mittels eines Browsers per Internet/Intranet (über das `http`-Protokoll) von einem Benutzer bedient werden können. Die Benutzung eines Browsers ist jedoch nicht zwingend: Webserver können auch auf Anfragen von anderen Programmen ihre Antworten senden. Sie sind plattformunabhängig und können durch den Browser auch von gängigen Smartphones, Tablets oder anderen mobilen Geräten aufgerufen werden.

Webapplikationen sind nicht mit Websites gleichzusetzen. Spricht man von den Webseiten einer Webapplikation, ist deren grafische Benutzeroberfläche gemeint, die durch die Anzeige des HTML-Codes entsteht, der von einem Webserver als Antwort an einen Browser verschickt wird. Im herkömmlichen Sinne versteht man unter Webseiten viele Einzelseiten, durch die ein Benutzer navigieren kann, um Texte, Bilder, Videos etc. abzurufen. Darum werden diese beiden Begriffe auch des Öfteren verwechselt.

Native Applikationen haben den Vorteil, dass sie auf alle Hardware-Funktionen von mobilen Geräten wie z.B. Kamera und Mikrofon zugreifen können und einfach zu installieren sind. Sie haben Performancevorteile, was sie für 3D-Programme und Spiele allgemein, die grafikintensiv sind, interessanter macht. Ähnliches gilt für rechenintensive Operationen. Dadurch, dass sie ihre Daten auf dem mobilen Gerät speichern, bleiben sie unabhängig von einer Netzwerkverbindung.

Weil Webapplikationen unabhängig von einem Betriebssystem sind, ist ihre Entwicklung i.d.R. günstiger als die Entwicklung einer nativen Applikation, die für jedes Betriebssystem und jeden Typ von mobilen Geräten einzeln entwickelt wer-

den muss. Sie können sehr schnell veröffentlicht werden und brauchen keinen Zulassungsprozess zu durchlaufen. Auch wenn der Aufwand für den Vertrieb und insbesondere für den Verkauf von nativen Apps geringer ist, entfällt für Web-Apps dabei die Provision an den Betreiber.

Im Band III des Übungsbuchs »Servlets und Java ServerPages« sind viele Beispiele von Webapplikationen zu finden. Dazu möchte ich jedoch anmerken, dass meine Übungsbücher zu Java, auf die ich des Öfteren im Laufe der Kapitel verweise, eine Hilfestellung sein können, aber durchaus keine Voraussetzung sind.

Heutzutage wird viel über die Vor- und Nachteile von nativen Applikationen gegenüber Webapplikationen gesprochen. Durch die Verwendung von HTML5 und CSS3 erhofft man sich nun, Funktionen, die früher nur für native Apps möglich waren, in Webapplikationen immer einfacher integrieren zu können. Mittlerweile stehen Entwicklern mehrere Frameworks zur Verfügung, die beim Erstellen von Applikationen mit nativem Layout und Verhalten eingesetzt werden können, darunter die bekannten `jQTouch` und `Sencha`. Diese werden auch eher, im Gegensatz zu den herkömmlichen Webapplikationen, in der Literatur als Web-Apps bezeichnet.

In Windows 8 werden Anwendungsprogramme allgemein als Apps bezeichnet.

Die wichtigsten Anbieter von Apps sind Google Play (ehemals Android Market) von Google, App Store von Apple, Windows Phone Store von Microsoft und Amazon App Shop von Amazon.

Mit dem Betriebssystem Windows 8 wird erwartet, dass parallel zu Geräten mit einem Android- bzw. Apple-iOS-Betriebssystem auch Windows-Smartphones und -Tablets einen wesentlichen Marktanteil beanspruchen werden.

1.2 Apps mit Android entwickeln

Dieses Buch soll eine schnelle und einfache Einführung in die Entwicklung von Android-Apps anhand von Beispielaufgaben für Programmierer mit Java-Kenntnissen geben. Fakt ist, dass jeder, der Java programmieren kann, auch Applikationen für das Android-Betriebssystem schreiben kann. Viele der Android-Lehrbücher helfen den Programmierern von Android-Apps dahin gehend, dass sie eine Einführung in Java (und parallel dazu für Eclipse) anbieten. Ich möchte darauf verzichten und auf die in den vorangegangenen Übungsbüchern erworbenen Java-Kenntnisse aufbauen, um die Entwicklung von Apps mit den hinzukommenden neuen Android-Klassen (mit und im Vergleich zu der Benutzung von Standard-Java-Klassen) darzustellen.

Android-Apps sind in Java geschrieben. Sie können die Java-APIs erst ab Java 5 nutzen und nur diejenigen, die unter `http://developer.android.com/refe-`

rence/packages.html beschrieben sind. So unterstützt Android z.B. weder AWT noch Swing und stellt dem Benutzer eine eigene kleinere Menge von UI(User Interface)-APIs zur Verfügung. Neben dem Java-SDK benötigt man für die Entwicklung das Android-SDK, das Bibliotheken mit neuen Java-Klassen und -Interfaces bereitstellt. Gleichzeitig stellt es dem Programmierer einen Emulator zum Erstellen und Testen von Apps auf dem PC zur Verfügung.

Android selbst wird im Android Developer's Guide unter http://developer.android.com/guide/components/index.html als ein »Software-Stack für mobile Geräte, das mehrere Software-Subsysteme beinhaltet, die gebraucht werden, um eine voll funktionsfähige Lösung zu liefern«, beschrieben.

Das Software-Stack besteht aus:

- einem Betriebssystem (eine modifizierte Version des Linux-Kernels)
- einem Applikation Framework, Bibliotheken und einem Runtime-System
- fertigen Applikationen wie Browser (Internet), Home, Phone (Telefon), Email (E-Mail), Gallery (Galerie) und Maps
- Die Laufzeitumgebung von Android besteht aus einem Bibliothekskern aus der Java-5-Implementation und der Dalvik Virtual Machine (»eine Nicht-JVM, die auf Prozessoren-Register basiert anstatt Stack-basiert zu arbeiten«).

Jede Android-App läuft standardmäßig in einem eigenen Linux-Prozess ab, der gestartet wird, wenn irgendeine Komponente von einer App gebraucht wird. Man spricht in diesem Zusammenhang auch von einem Sandbox-Prinzip. Dieser Prozess wird beendet, wenn dies nicht mehr der Fall ist, um anderen Applikationen die nicht mehr benötigten Systemressourcen zur Verfügung zu stellen. Dadurch, dass sich Applikationen keinen gemeinsamen Speicher teilen, wird ihre Sicherheit und Verfügbarkeit unterstützt. Dies erhöht jedoch nicht unerheblich den Anteil an benötigten Betriebssystem-Ressourcen.

In der Nicht-Java-VM von Dalvik kann auch kein Java-Code ablaufen. Darum konvertiert Android die Java-Klassendateien, die erstmals mit dem Java-Compiler javac übersetzt werden, in ein sogenanntes Dalvik-Executable-Format (DEX), das in der Dalvik-VM ablauffähig ist.

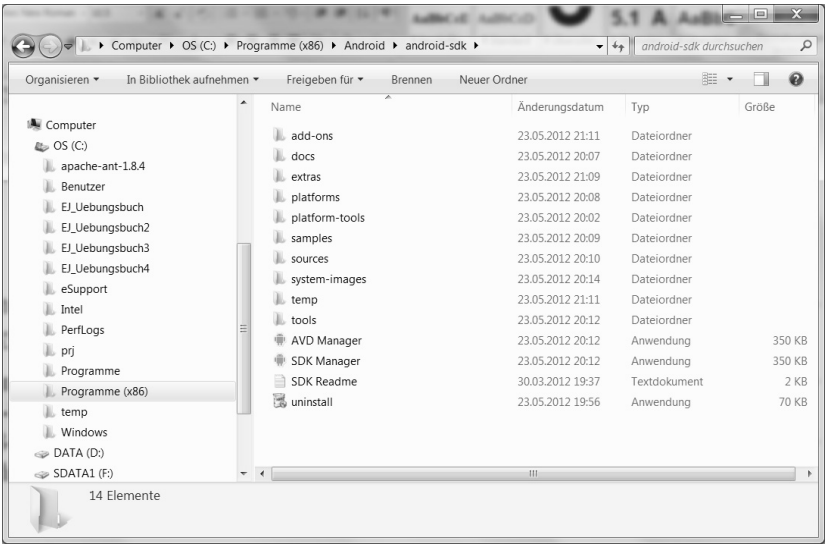
1.3 Das Android-SDK und eine Android-Plattform installieren

Das aktuelle Release des Android-SDK kann für die gewünschte Plattform von http://developer.android.com/sdk/index.html runtergeladen werden.

Für Windows (es werden die Betriebssysteme Windows XP (32-bit), Vista (32- oder 64-bit) und Windows 7 (32- oder 64-bit) unterstützt) kann das Archiv android-

`sdk_r18-windows.zip` oder die Setup-Datei `installer_r18-windows.exe` (von Android empfohlen) für die Installation des Android-SDK-Release-18 gebraucht werden.

Nach der Installation in einem von Ihnen ausgewählten Verzeichnis finden Sie in einem Unterverzeichnis `android-sdk` die verschiedenen Komponenten des Android-SDK:



Falls Sie alle Komponenten installiert haben, sehen die angelegten Unterverzeichnisse mit diesem Release wie oben aus. Weil ich beim Entwickeln von Apps auf die dafür erforderlichen Tools zurückkommen werde, möchte ich an dieser Stelle genauer nur auf das Verzeichnis `docs` aufmerksam machen. In seinem Unterverzeichnis `references` ist eine Datei `index.html` hinterlegt, die ein Inhaltsverzeichnis darstellt, aus dem auf die vollständige Android-Dokumentation der API-Referenz verzweigt werden kann. Die Android-API-Dokumentation ist analog der Java-API-Dokumentation aufgebaut. Durch die Auswahl eines bestimmten Paketnamens (packages) können alle darin enthaltenen Klassen und Interfaces angezeigt werden. Die Auswahl einer Klasse oder eines Interface führt uns zu der Beschreibung von deren Feldern und Methoden.

Diese Installation reicht nicht aus, um Android-Apps zu entwickeln. Dafür muss zusätzlich eine sogenannte Android-Plattform installiert werden.

Starten Sie dazu das SDK-Manager-Tool aus dem Verzeichnis `android-sdk` durch einen Doppelklick auf die Datei `SDK Manager.exe` oder über einen Aufruf auf

Kommandozeilenebene. Es wird eine Dialog-Box geöffnet, mit einer größeren Auswahl an Komponenten (eine Package-Liste), die zur Installation bereitstehen. Davon sind einige mit einem kleinen Häkchen vorgemerkt. Installieren Sie am besten erstmals nur die aktuellste Version, weil diese die neuesten Features bereitstellt. Es können jedoch, falls gewünscht, auch mehrere Plattformen gleichzeitig installiert werden, auf die beim Anlegen von Android-Projekten gezielt Bezug genommen werden kann. Deaktivieren Sie die nicht gewünschten Auswahlmöglichkeiten und klicken Sie auf den `Install`-Button.

Nach der Installation werden alle installierten Packages für die von Ihnen ausgewählten Plattformen angezeigt. Ich habe mich zu Beginn für die zum Zeitpunkt der Redaktion dieses Buchs gerade aktuelle Version 4.0.3 API 15 entschieden. Bei der Durchführung von Korrekturen wurde für Tests die Version 4.2.2 API 17 benutzt (siehe dazu das Unterkapitel 3.11).

Noch eine Anmerkung, die hilfreich sein kann: Sollte bei der Installation des Android-SDK unter Windows 7 (64-Bit-Version) das Java-SDK nicht erkannt werden, kann dazu in der Systemsteuerung wahlweise systemweit oder nur für den aktuell angemeldeten User die Systemvariable JAVA_HOME auf den Pfad gesetzt werden, wo sich das JDK befindet (wie z.B.: `C:\Programme\Java\jdk1.7.0_17`). Wenn dies nicht funktioniert, sollte man die Unix-Schreibweise ausprobieren (`'/'` anstelle von `'\'`). Wichtig ist, dass der Benutzer sich ab- und dann wieder anmeldet, damit die neue Systemvariable dem Betriebssystem bekannt gemacht wird. Ein kompletter Neustart des PC ist nicht erforderlich.

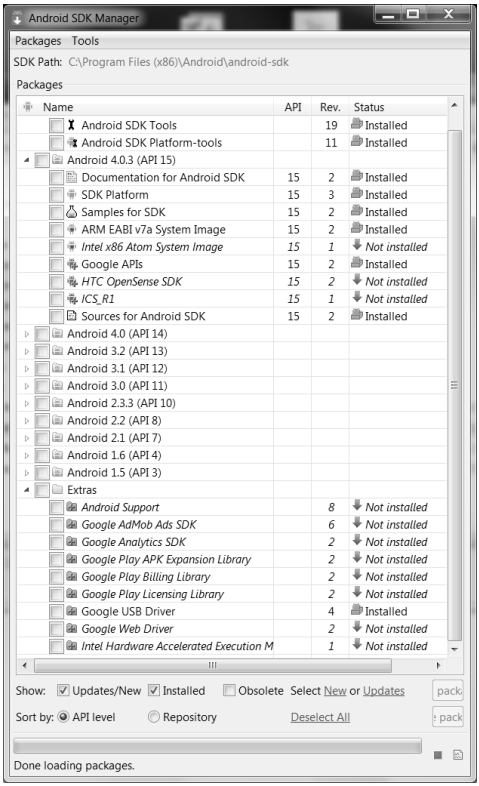
Nach der Installation des SDK und einer Plattform für Android können wir auch gleich die Entwicklung von Apps angehen. Zum Test benutzen wir den Android-Emulator, ein Programm, das ein Gerät (Device) mit Android-Betriebssystem auf dem PC emulieren kann. Dies hat den Vorteil, dass die entwickelten Applikationen nicht nur für ein bestimmtes Gerät mit einer festgelegten Hardware getestet werden können, sondern für eine größere Vielfalt von Devices. Falls ein anderer Geräte-Typ getestet werden soll, muss nur die Hardwareabstraktionsschicht, die den Linux-Kernel von der restlichen Software trennt und über die verschiedene Programme auf die Hardware eines mobilen Geräts zugreifen, angepasst werden.

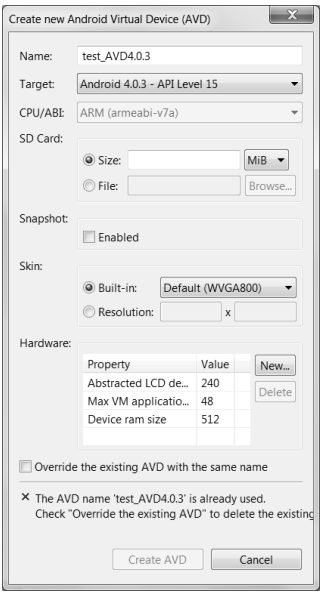
Die Beschreibung des Geräts, das vom Emulator simuliert wird, erfolgt durch einen Descriptor, der als AVD (Android Virtual Device) bezeichnet wird und die Fähigkeiten des virtuellen emulierten Geräts definiert (wie z.B. Bildschirmgröße und Auflösung, Größe des Hauptspeichers und der SD-Karte etc.).

Für das Anlegen eines AVD wird analog zum vorangegangenen Vorgang der AVD-Manager mit der Datei `AVD Manager.exe` gestartet (für ältere Plattformen existiert für beide Manager eine gemeinsame `.exe`-Datei). Beim Betätigen des `New`-Buttons wird eine Dialog-Box, in der ein Name für das AVD (ich habe mich für `test_AVD4.0.3` entschieden) und die Version der benutzten Android-Plattform einge-

geben werden müssen, angezeigt. Die Hardware-Eigenschaften für das virtuelle Device werden automatisch der ausgewählten Plattform angepasst und in den zugehörigen Feldern der Box eingetragen.

Bevor eine App getestet und installiert werden kann, muss ein AVD gestartet werden. Dies geschieht über dessen Auswahl auf dem Device-Panel mithilfe des `Start`-Buttons. Danach erscheint die Launch-Options-Dialog-Box, in der der `Launch`-Button betätigt werden muss, um den Emulator über das AVD zu starten. Dies kann einige Minuten dauern und wird durch die Anzeige des AVD-Windows beendet. Dieses Fenster zeigt als Erstes das Android-Logo auf einem schwarzen Hintergrund auf der linken Seite und rechts die sogenannten »phone controls« und eine Tastatur. Wird `Menu` ausgewählt, erscheint links der »home screen« des Android-Devices.





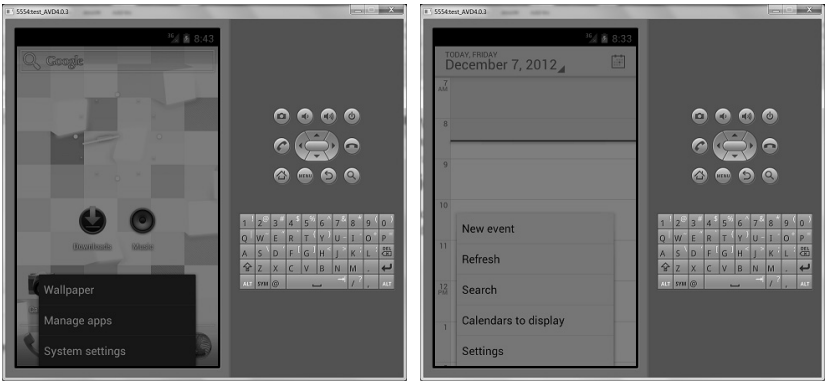
Das AVD-Fenster zeigt den Text `5554: test_AVD4.0.3` in seiner Titelleiste an. Android unterstützt das Starten von 16 parallel laufenden AVDs, die über eine Konsolen-Port-Nummer beginnend mit 5554 durchnummeriert werden.

Der Home-Bildschirm ist laut Android-Dokumentation eine spezielle App, die als Startpunkt für das Device und Bildschirm für andere Applikationen dient. Sein Hintergrund kann abgeändert werden. Betätigen Sie dazu erneut die `Menu`-Taste, wählen Sie die Option `wallpaper` im angezeigten Pop-up-Menü und im darauf folgenden einen neuen Hintergrund.

Über dem Android-Home-Bildschirm liegt eine Statuszeile (die die aktuelle Uhrzeit und den Status für den Batterieverbrauch anzeigt) und darunter liegt der App-Launcher. Über dessen mittleren (gepunkteten) Button können die Icons für alle installierten Apps angezeigt werden. Durch die Auswahl eines Icons kann die gewünschte App gestartet werden.

Über die Phone-Control-Pfeile kann zwischen unsichtbaren Panels, in die der Home-Screen aufgeteilt ist, hin und her geschaltet werden. Außerdem sind darin Schaltflächen zum Auslösen der Kamera, zum Simulieren des Abnehmens und Ablegens des Telefons und zur Änderung der Lautstärke enthalten.

Mit der `Menu`-Taste kann nicht nur zum Home-Bildschirm, sondern auch zu jedem Bildschirm einer laufenden App ein Kontextmenü mit verschiedenen Auswahlmöglichkeiten angezeigt werden:



Mithilfe des `Back`(Rückkehr)-Buttons (geschwungener Pfeil in der Anzeige) kann zu einem vorher angezeigten Bildschirm zurückverzweigt werden (siehe dazu die genauere Beschreibung aus dem Unterkapitel 1.4).

Sie können mit dem AVD interagieren, indem Sie die Maus für das »Berühren« des Bildschirms benutzen und die Tastatur zum Drücken der Device-Tasten, die auf die Tasten Ihres Entwicklungs-PCs abgebildet sind.

1.4 Die Architektur einer Android-App

Die Android-Apps unterscheiden sich sowohl von Java-Applikationen, die auf dem Desktop ablaufen, als auch von Windows- und Web-Applikationen. Sie bestehen aus mehreren Komponenten: Activities, Intents, Content Provider, Broadcast Receiver und Services.

Während Windows-Applikationen standardmäßig ein Hauptfenster besitzen, in dem ein Benutzer mehrere Aktionen durchführen kann, besteht eine App aus einer oder mehreren Bildschirmseiten, die jede einer bestimmten Activity-Komponente zugeordnet ist.

Der Komponenten-orientierte Aufbau einer App führt dazu, dass ihre Komponenten auch von anderen Apps benutzt werden können, und reduziert so den globalen Speicherbedarf für Applikationen, was für Geräte mit limitiertem Speicher sehr wichtig sein kann. Über diese Eigenschaft können eigene Applikationen auch die Komponenten von Standard-Android-Applikationen nutzen.

Activities werden als Unterklassen der Klasse `android.app.Activity`, die selbst die Klasse `android.content.Context` erweitert, implementiert. Diese Unterklassen überschreiben verschiedene Callback-(Rückruf-)Methoden der Oberklasse

(Basisklasse), die von Android während des Lebenszyklus einer Activity automatisch aufgerufen werden.

Eine Activity muss nicht alle 7 Methoden ihrer Oberklasse, die den kompletten Lebenszyklus einer Activity ausmachen, überschreiben. Alle Activities müssen jedoch die `onCreate()`-Methode überschreiben, die aufgerufen wird, wenn eine Activity erzeugt wird.

- Die `onCreate()`-Methode bekommt eine `android.os.Bundle`-Referenz übergeben, die auf ein `Bundle`-Objekt zeigt, das einen vorangegangenen Status der Activity enthält, wenn dieser abgefangen wurde, und ansonsten `null`. Sie wird benutzt, um die Benutzeroberfläche zu definieren, Hintergrund-Threads, falls benötigt, zu erzeugen und globale Initialisierungen durchzuführen.
- Die `onStart()`-Methode wird aufgerufen, nachdem eine Activity erzeugt wurde oder wenn sie vorher für den Benutzer nicht sichtbar war und wieder angezeigt wird (wenn z.B. mittels der »Zurück«-Taste eine Rückkehr in das Launch-Fenster erfolgte).
- `onRestart()` wird aufgerufen, wenn eine noch vorhandene Activity gestoppt wurde, bevor sie eventuell neu gestartet wird (wenn z.B. der Benutzer die App im Launcher wieder angetippt hat).
- `onResume()` wird aufgerufen, bevor die Activity anfängt, mit dem User zu interagieren. Die Activity bekommt zu diesem Zeitpunkt den Fokus und kann auf Benutzereingaben reagieren.
- Die `onPause()`-Methode wird aufgerufen, wenn Android dabei ist, eine andere Activity aufzurufen, um nicht gespeicherte Änderungen festzuhalten, eine Animation zu stoppen etc. Die Activity ist in diesem Fall nur teilweise sichtbar, weil sie von einer anderen Activity bedeckt wird.
- `onStop()`-wird aufgerufen, wenn eine Activity für den Benutzer nicht mehr sichtbar ist und vollständig in den Hintergrund tritt. Dies kann der Fall sein, wenn eine Activity zerstört wurde oder eine andere, die gestartet wurde, diese verdeckt.
- Die `onDestroy()`-Methode wird aufgerufen, bevor eine Activity zerstört wird, es sei denn, ein Speicherengpass ist aufgetreten und Android wird gezwungen, den Activity-Prozess zu beenden.

Für das Beenden einer Activity in einem Programm kann die Methode `finish()` aufgerufen werden. Die Android-Dokumentation rät jedoch davon ab, weil im Normallfall die `Back`-Taste des mobilen Geräts verwendet wird, um zu einer vorangegangenen Activity zurückzukehren oder um eine App zu verlassen.

Der Lebenszyklus einer Activity (wird mit den Aufgaben 2.1 und 2.3 näher dargestellt) beinhaltet drei Arten von inneren Schleifen:

- Der »komplette Lebenszyklus« einer Activity, in dem alle ihre Zustände durchlaufen werden, wird ab dem ersten Aufruf von `onCreate()` bis zu einem Aufruf von `onDestroy()` definiert.
- Ein »sichtbarer Lebenszyklus« bezeichnet die Zeit zwischen einem Aufruf von `onStart()` und dem korrespondierenden `onStop()`-Aufruf. Während dieser Zeit kann der Benutzer die Bildschirmanzeige sehen, aber dadurch, dass diese nicht im Vordergrund steht, ist keine Interaktion mit dem Benutzer möglich.
- Der sogenannte »Vordergrund-Lebenszyklus« wird zwischen einem Aufruf von `onResume()` und dem korrespondierenden `onPause()`-Aufruf definiert. Die Activity befindet sich während dieser Zeit im Vordergrund von allen Bildschirmanzeigen der anderen Activities und kann mit dem Benutzer interagieren.

Die zweite Art von Komponenten, die eine wichtige Rolle im Ablauf von Android-Applikationen spielen, sind die Intent-Komponenten.

Generell können Intents als Messages (Systemnachrichten) betrachtet werden, die dazu dienen, das Betriebssystem über das Eintreten bestimmter Ereignisse zu informieren. Im Fall von Activities können sie eine Operation, die ausgeführt werden soll, beschreiben, wie z.B. das Senden einer E-Mail oder die Auswahl eines Fotos. Im Fall von Broadcasts können sie Beschreibungen von externen Events liefern, die aufgetreten sind (wie z.B. das Aktivieren einer Kamera).

Derartige Nachrichten (auch als »Botschaften« in der Literatur bezeichnet) werden als Instanzen der Klasse `android.content.Intent` erstellt und als Kombination von mehreren Arten von Informationen, die über eine eigene Bezeichnung verfügen, vermittelt:

- `Action`: Über einen String wird die Aktion, die ausgeführt werden soll, benannt und mit der Methode `setAction()` gesetzt. Sie können mithilfe von Intent-Konstanten beschrieben werden, aber auch mit selbstdefinierten Action-Strings.
- `Component name`: Mit einem String wird wahlfrei der vollqualifizierte Name einer Komponenten-Klasse spezifiziert, die den Intent behandeln soll.
- `Category`: kann zusätzlich oder alternativ zum Komponentennamen angegeben werden und enthält zusätzliche Informationen zu der Komponente, die ausgeführt werden soll. Diese Informationen können mit `addCategory()` gesetzt werden.
- `Data`: spezifiziert die zu verwendenden Daten über einen URI(Uniform Resource Identifier)-Eintrag. URIs bestehen aus einer Folge von Zeichen, die zur

Identifizierung einer Ressource im Internet dienen. URIs können vom Typ URL (Uniform Resource Locator) und URN (Uniform Resource Name) sein. Während URLs die Location einer Ressource und die unterstützten Zugriffsarten beinhalten (wie z.B. `http://de.wikipedia.org/wiki/Uniform_Resource_Identifier`), identifizieren URNs eine Ressource mittels eines Namens (wie z.B. `ISBN:978-3-8266-9203-1`). Mit anderen Worten: Während eine URL eine Methode zum Auffinden einer Ressource liefert, definiert eine URN ihre Identität. Weil im Nachfolgenden des Öfteren von URIs und URLs die Rede sein wird, möchte ich bereits an dieser Stelle auf beide Begriffe hinweisen, mit der Anmerkung, dass diese oft im Sprachgebrauch gleichbedeutend verwendet werden.

- `Extras`: beinhalten zusätzliche Daten, die als Schlüssel-Wert-Paare übergeben werden können.
- `Flags`: informieren Android mithilfe von Bit-Kombinationen, wie eine Activity gestartet werden soll. Diese werden als Konstanten in der `Intent`-Klasse definiert.
- `Type`: gibt den MIME(Multipurpose Internet Mail Extensions)-Typ der Intent-Daten an. Dieser wird im Normallfall von Android selbst inferiert, ansonsten kann er mit der Methode `setType()` gesetzt werden.

Intents sind in erster Linie dazu da, eine Verbindung zwischen Activities, Content Provider, Broadcast Receiver und Services herzustellen.

Wenn sie verwendet werden, um eine Activity zu aktivieren, wird dem Benutzer ein neuer Bildschirm auf dem mobilen Gerät angezeigt. In diesem Fall werden Intents mit der Methode `startActivity()` oder `startActivityforResult()` der `Activity`-Klasse gestartet. In beiden Methoden wird als Argument die `Intent`-Instanz übergeben. Die zweite Methode definiert einen zusätzlichen `int`-Parameter, in dem ein Code übergeben wird, der zurückgegeben werden soll, sobald die gestartete Activity wieder beendet wurde (so kann dieser entsprechend ausgewertet werden).

Wird im Intent der Name der Komponente, die gestartet werden soll, angegeben, sprechen wir von expliziten Intents. Implizite Intents, in denen dem Komponentennamen kein Wert zugewiesen wird, geben nur die Eigenschaften an, anhand derer die Komponente[n] identifiziert werden soll[en]. In diesem Fall wählt Android die Zielkomponente anhand von Intent-Filtern, die angeben, für welche Kombination von `Action`, `Category` und `Data` diese gestartet werden kann.

Implizite Intents werden typischerweise verwendet, um Activities, die zu anderen Apps gehören, zu starten. Explizite Intents werden eher innerhalb derselben App, falls diese mehrere Activities besitzt, eingesetzt, um diese untereinander aufzurufen.

Android besitzt eine ganze Reihe von impliziten Standard-Intents, die über eine `Action`-Art erzeugt werden und dem Anwender ein View für das Durchführen der Aktion bereitstellen. So z.B. wird mittels Angabe der Konstanten `Intent.ACTION_SET_WALLPAPER` im Konstruktor der Klasse `Intent` ein Dialogfenster für die Auswahl eines Hintergrunds für den Home-Bildschirm des mobilen Geräts angezeigt oder mit der Übergabe von `Intent.ACTION_DIAL` im Konstruktoraufruf ein View für die Eingabe einer Telefonnummer zur Verfügung gestellt.

Sowohl expliziten als auch impliziten Intents können zusätzliche Daten hinzugefügt werden. Mit mehreren überladenen `putExtra()`-Methoden der Klasse `Intent` kann die Komponente, die den Intent erzeugt, Daten in Form von Schlüssel-Wert-Paaren in sogenannte Extras schreiben, um diese an die aufgerufene Komponente zu überreichen. Die Schlüssel derartiger Abbildungen (auch Maps genannt) werden standardmäßig als Strings definiert. Als Werte können primitive Datentypen (wie `int` und `long`), `String`, `Bundle` etc. benutzt werden.

Diese Daten können mit der Methode `getExtras()`, die eine `Bundle`-Instanz liefert, zurückgeholt werden. Mit Methoden wie `getInt()`, `getLong()`, `getIntArray()` etc., in denen ein Schlüssel übergeben wird, können die diesen zugeordneten Werte ermittelt werden.

So kann z.B. an alle Activities, die sich für das Senden von Daten mittels eines `Intent.ACTION_SEND`-Intents registriert haben (und damit gestartet werden), mit einem beliebigen Schlüssel, oder unter Benutzung eines von Android vordefinierten Schlüssels für diesen Intent, ein Text übergeben werden:

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(android.content.Intent.EXTRA_TEXT, "Hier sind Ihre News!");
startActivity(intent);
```

Vordefinierte Schlüssel müssen immer den zugehörigen Paketnamen vorangestellt bekommen. Die Komponenten, die diesen Intent empfangen, können wiederum mit:

```
Bundle extras = getIntent().getExtras();
String value = extras.getString(Intent.EXTRA_TEXT);
```

den hinzugefügten Text aus den Extras ermitteln.

Eine beliebige Activity kann für den `ACTION_SEND`-Intent mit dem Mime-Typ `text/plain` wie folgt in der AndroidManifest-Datei registriert werden:

```
<activity
   android:name="SendActivity"
   android:label="@string/app_name" >
   <intent-filter>
      <action android:name="android.intent.action.SEND" />
```

```
      <category android:name="android.intent.category.DEFAULT" />
      <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

Eine Komponente, die keinen Intent-Filter definiert, kann nur über explizite Intents erreicht werden. Mit mehreren Aufgaben aus den Kapiteln 2 und 3 werden weitere Beispiele zur Nutzung von Intents, insbesondere in der Kommunikation von Activities, geliefert.

Android bezeichnet eine Folge von zusammenhängenden Activities als Task und liefert einen sogenannten Activity-Stack, in dem diese in einer bestimmten Reihenfolge, entsprechend des Zeitpunkts ihrer Aufrufe, abgelegt werden. Die Activity, die den Task startet, wird auch als `root`-Activity bezeichnet und ist typischerweise die Activity, die vom Benutzer über den App-Launcher des mobilen Gerätes ausgewählt wird. (Darum wird standardmäßig der Name dieser Activity auch als Name für die Applikation gewählt.) Sie wird über einen Intent gestartet, der versendet wird, wenn der Benutzer auf das Symbol der App getippt hat.

Die gerade laufende Activity ist immer die erste im Stack. Wenn der Benutzer die »Zurück«-Taste betätigt, wird die aktuelle Activity aus dem Activity-Stack entfernt und die vorangegangene Activity wird als laufende Activity ihre Aktivitäten fortsetzen. Die Activities werden nie im Stack neu geordnet, sondern einfach wieder hinzugefügt oder entfernt. Dieser Prozess wird so lange durchgeführt, bis der Benutzer mit der »Zurück«-Taste in den Home-Bildschirm von Android oder eine andere Activity, die vor dem Beginn des zugehörigen Tasks lief, zurückkehrt. Indem die `destroy()`-Methode der Start-Activity aufgerufen wird, wird die Applikation beendet. Wenn alle Activities aus dem Stack gelöscht wurden, ist auch der Task beendet.

Content Provider werden eingesetzt, um Daten, die von einer oder mehreren Applikationen verwendet werden, zu veröffentlichen und zu verwalten. Diese Daten können z.B. im Dateisystem von Android oder in seiner `SQLite`-Datenbank gespeichert sein. Content Provider werden mithilfe der Klasse `android.content.ContentProvider` implementiert.

Fragments können ab der Version 3.0 (API-Level 11) für Tablets und ab der Version 4.0 für Smartphones von Android benutzt werden, um den Code umfangreicher Activities aufzuteilen. Sie verfügen über eine eigene Benutzeroberfläche, einen eigenen Lebenszyklus und werden durch Instanzen der Klasse `android.content.Fragment` repräsentiert.

Die anderen vorher aufgezählten Komponenten von Android-Apps werden in dieser kurzen Zusammenfassung keine besondere Rolle spielen. Der Vollständigkeit halber werden an dieser Stelle auch deren Zuständigkeiten erwähnt.

Services sind Komponenten, die im Hintergrund für eine unbestimmte Zeit ablaufen und über keine eigene Benutzeroberfläche verfügen. Damit können länger dauernde Operationen parallel zur laufenden App ausgeführt werden, wie z.B. das Herunterladen von Multimediadaten aus dem Internet. Zum anderen können Services dazu benutzt werden, Funktionalitäten anderen Applikationen zur Verfügung zu stellen (im Gegensatz zu Content Provider, die Daten applikationsübergreifend bereitstellen). Weil Applikationen und Services in unterschiedlichen Prozessen laufen können, wird dazu die IPC (Inter Process Communication) benutzt. Benutzer-Services werden mithilfe der Klasse `android.app.Services` implementiert. System-Services werden im Unterkapitel 3.3 kurz beschrieben.

Broadcast Receiver werden als Objekte der Klasse `android.content.BroadcastReceiver` erzeugt. Diese können auf die internen Ereignisse des Android-Systems reagieren und dabei Nachrichten in der System-Statusleiste anzeigen.

1.5 Das Android-Projekt

Viele der Android-Lehrbücher arbeiten von vornherein mit Eclipse. Dazu muss das zugehörige Android-Plug-in installiert werden und Eclipse entsprechend konfiguriert werden. Das Entwickeln einer App beginnt in diesem Fall mit dem Anlegen eines Projekts in der Entwicklungsumgebung. Mit einem Android-Wizzard, der mit dem Android-Plug-in installiert wird, wird der Programmierer schrittweise zum Anlegen des Projekts geführt. Mit der Installation des ADT(Android Developer Tools)-Plug-ins für Eclipse werden der Entwicklungsumgebung mehrere Android-spezifische Tools hinzugefügt, die beim Design, Entwickeln, Debuggen und Veröffentlichen von Applikationen eingesetzt werden können.

Wir wollen auch diesmal den Weg ohne eine Entwicklungsumgebung gehen, um die einzelnen Schritte beim Anlegen eines Android-Projekts besser verfolgen zu können, und das Apache-Tool `ant` zur Hilfe holen. Der kompilierte Java-Code wird damit in das schon erwähnte DEX-Format umgesetzt und zusammen mit allen anderen erforderlichen Daten und Projekt-Ressourcen in eine sogenannte APK(App PacKage)-Datei mit dem Suffix `.apk` gebunden.

Dazu muss die Ant-1.8-Version (oder höher) installiert werden. Sie können eine dieser Versionen von der Website `http://ant.apache.org` herunterladen.

Ungestört von dem gesamten Overhead, das eine Entwicklungsumgebung mit sich bringt, können damit die flexiblen und schnellen Kommandozeilen-Programme des Android-SDK benutzt werden, um effizient arbeiten zu können. Selbstverständlich können Sie für den Fall, dass Sie die Arbeit mit einer Entwicklungsumgebung bevorzugen, die hier bereitgestellten Dateien mit den Ressourcen-Editoren und dem UI-Designer weiter bearbeiten und in Ihre Apps importieren.

Der erste Schritt beim Anlegen eines Android-Projekts besteht in der Festlegung eines Projektpfads, eines Namens für das Projekt und für die Start-Activity der App und einem Paketnamen für Java-Klassen. Dem kann der Aufruf des `android`-Tools (im Unterverzeichnis `tools` von `android-sdk` abgelegt) folgen. Dazu ist folgende Syntax zu benutzen: `android create project –t[arget] target_ID –n[ame] projekt_name –p[ath] projekt_pfad] –a[ctivity] activity-name –p[ackage] paket-name`, wobei gilt:

- Die `–target`-Option ist ein `int`-Wert, der die Android-Plattform identifiziert und mit `android list targets` ermittelt werden kann. Wenn nur eine Plattform installiert ist, gibt dieser Aufruf den Wert 1 zurück.
- Die Option `–name` ist die einzige wahlfreie Option und liefert den Namen des Projekts. Er wird, falls angegeben, als Name für die `.apk`-Datei angenommen.
- Die `–path`-Option gibt den Pfadnamen des Projekts an. Der Projektpfad wird neu angelegt, falls er noch nicht existiert.
- Die `–activity`-Option spezifiziert den Namen der Start-Activity (die als erste ausgeführt werden soll). Wurde kein Projektname angegeben, dient dieser als Name für die `.apk`-Datei.
- Die Option `–package` definiert den Namen des Pakets, in dem die Java-Klassen einer App abgelegt werden, und unterliegt den Regeln für Package-Definitionen aus der Java-Spezifikation.

Durch das Ausführen dieses Kommandos werden mehrere Verzeichnisse angelegt und Dateien erstellt.

Um über die Dateien, die ein Projekt bilden, konkret sprechen zu können, legen Sie mit `android create project –t 1 –p C:\androidprj\apps\HalloApp –a HalloApp -k jung.j8hallo` das erste Projekt an und betrachten Sie das dabei erstellte Code-Gerüst.

Unter dem Verzeichnis `C:\androidprj\apps\HalloApp` finden Sie die Datei `AndroidManifest.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
   android" package="jung.j8hallo"
   android:versionCode="1"
   android:versionName="1.0">
   <application android:label="@string/app_name"
            android:icon="@drawable/ic_launcher">
     <activity android:name="HalloApp"
            android:label="@string/app_name">
       <intent-filter>
```

```
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.
                                              LAUNCHER" />
        </intent-filter>
      </activity>
    </application>
</manifest>
```

XML-Dateien spielen sowohl in Webapplikationen als auch in nativen Applikationen eine wichtige Rolle. Weil sie über einen einheitlichen Aufbau verfügen und in Android-Apps meistens ein Anreihen von View-Definitionen beinhalten, reicht ein Betrachten der standardmäßig angelegten Dateien, um sie zu erweitern bzw. neue erstellen zu können (Näheres über XML-Dateien kann auch im Band III des Übungsbuchs »Servlets und JavaServer Pages« Kapitel 2, Seiten 27 und 219 gelesen werden).

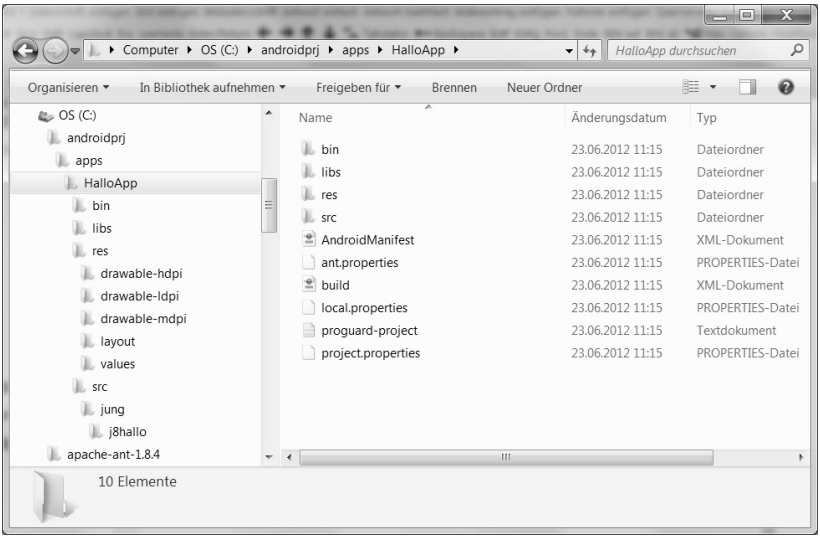
Durch die AndroidManifest-Datei wird die App mit ihren Komponenten dem Android-System bekanntgegeben. Sie erinnert an die Deployment-Descriptoren aus dem Java-EE-Umfeld.

Das `manifest`-Tag spezifiziert im `android`-Attribut den Namensraum von Android und im `package`-Attribut den Paketnamen der App (`jung.j8hallo`), der wie schon erwähnt, eindeutig für jede App definiert werden muss.

Das innerhalb von `manifest` definierte `application`-Tag spezifiziert mit den Attributen `label` und `icon` die vom Android-Gerät benutzten Ressourcen für Label und Icon, um die App im Launcher anzuzeigen. Die Ressourcen einer App werden generell durch das Präfix `@`, gefolgt von einer Category wie z.B. `string` oder `drawable`, gekennzeichnet.

- Innerhalb von `application` befindet sich ein `activity`-Tag, das eine Activity-Komponente identifiziert, indem im Attribut `android:name` der Name der Java-Klasse, die die Activity implementiert, angegeben wird. Mit den Attributen `android:label` und `android:icon` kann das `activity`-Tag Label und Icon der App überschreiben.
- Das innerhalb von `activity` definierte `intent-filter`-Element beschreibt die Rolle der Activity-Komponente. Das `action`-Element signalisiert über den Eintrag: `android:name="android.intent.action.MAIN"`, dass es sich dabei um die Start-Activity handelt, und der Eintrag aus dem `category`-Element `android:name="android.intent.category"` spezifiziert, dass die Activity im App-Launcher angezeigt werden soll, aus dem der Benutzer sie starten kann.

Des Weiteren befinden sich im Verzeichnis `HalloApp` eine `build.xml`-Datei, mehrere Properties-Dateien und mehrere Unterverzeichnisse, die als Speichergerüst für die Java-Klassen, `.class`-Dateien, `.apk`-Datei und Ressourcen der App dienen werden:



Wir werden diese mit den nachfolgenden Übungsaufgaben ergänzen und auf die dazugehörigen Grundlagen immer wieder zurückkommen.

1.6 Layout-, Ressourcen- und Klassendateien

Das Layout einer App zeigt, wie die Benutzeroberfläche auszusehen hat. In Android ist es üblich, das Layout über XML-Dateien zu definieren, die im Unterverzeichnis `layout` von `res` abgelegt werden. Darin werden die Komponenten, die die Oberfläche ausmachen und in Android als Views benannt werden, beschrieben. Layouts sind eigentlich nur eine Ansammlung von Eigenschaften und Anordnungsvorschriften für diese Komponenten.

View-Elemente (auch UI-Elemente genannt), die mit dem Anwender interagieren können, werden in Android auch als Widgets bezeichnet. Wie immer werden wir nicht auf alle damit verbundenen Details eingehen, sondern exemplarisch die eine oder andere Komponente für Illustrationszwecke benutzen. Eine ausführliche Auflistung aller Felder und Methoden der zugehörigen Klassen finden Sie unter `http://developer.android.com/reference/android/widget/package-summary.html`.

Views sind Instanzen der Klasse `android.view.View` und ihrer Unterklassen. Sie sind den AWT- und Swing-Komponenten aus Java ähnlich und können wie diese über die Konstruktoren von Klassen mit dem `new`-Operator zur Laufzeit erzeugt werden. Davon wird in Android jedoch eher nicht Gebrauch gemacht, sondern

beim Erstellen von Komponenten auf Layoutdateien zurückgegriffen. Aus den XML-Definitionen werden in der Entwicklungsphase vom Ressourcen-Compiler (`aapt`-Tool) `View`-Objekte erzeugt (siehe dazu die zwei unterschiedlichen Klassendefinitionen aus der Aufgabe 1.1).

In beiden Fällen werden diese Komponenten mit `setContentView()` in einem Java-Programm seiner Oberfläche hinzugefügt. Das zu verwendende Layout wird als Argument in dieser Methode übergeben.

Das `aapt`-Tool befindet sich im Verzeichnis `android-sdk\platform-tools`, aus dem es auch direkt gestartet werden kann. Ein weiteres wichtiges Verzeichnis, das bei der Installation des SDK unter Windows angelegt wird, ist das Verzeichnis `tools`.

Views werden ihrerseits in View-Gruppen zusammengefasst, die durch Instanzen der Klasse `android.view.ViewGroup` repräsentiert werden und Java-Containern ähnlich sind. Die Klasse `ViewGroup` ist wie auch alle Android-Widget-Klassen, darunter `Button`, `ImageButton`, `RadioButton`, `EditText`, `TextView`, `ImageView`, `WebView`, `ProgressBar`, `Spinner` und `ListView`, von der Klasse `View` abgeleitet. Sie ist die Oberklasse von `LinearLayout`, `RelativeLayout`, `TableLayout` etc. Android kennt eine ganze Reihe von Layout-Views, die andere View-Elemente aufnehmen können. Sollen die Komponenten einer Benutzeroberfläche einfach nur nebeneinander oder untereinander angereiht werden, wird ein LinearLayout gebraucht. Die detaillierte Dokumentation zu Layout-Definitionen finden Sie unter `http://developer.android.com/guide/topics/ui/declaring-layout.html`. In den nachfolgenden Aufgaben werden die wichtigsten davon benutzt.

Für jede `View`-Unterklasse existiert immer ein passender Eintrag in der `.xml`-Datei, die die Oberfläche einer App definiert.

Um den Aufbau von Layoutdateien einzuüben, sehen wir uns die vom Android-System beim Anlegen eines Projekts standardmäßig erstellte Datei `main.xml` aus dem `res\layout`-Ordner näher an:

```
<?xml version="1.0" encoding="UTF-8"?>
   <LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android">
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:orientation="vertical"
     <TextView
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="Hello World, HalloApp"/>
   </LinearLayout>
```

Das Element, in dem die Beschreibung eines LinearLayouts vorgenommen wird, besitzt mehrere Attribute, die seine eigene Ausrichtung und die eingebetteter Elemente definieren:

- `android:orientation` kann die Werte `"horizontal"` (der Standard) und `"vertical"` aufnehmen und zeigt, wie die View-Elemente angeordnet werden sollen.
- `android:layout_height` definiert die Höhe des Layouts und kann die Werte `"fill_parent"` (nimmt die Größe ihres Vorfahrens an) und `"wrap_content"` (nimmt die erforderliche Größe für die Anzeige der Komponente an) zugewiesen bekommen. Ab Android 2.2 wurde `"fill_parent"` durch `"match_parent"` ersetzt, wird aber weiter unterstützt. Auch fest vorgegebene Größen, wie z.B. "150 dp", sind zugelassen.
- `android:layout_width` definiert die gleichen Eigenschaften für die Weite des Layouts.

Innerhalb des `LinearLayout`-Tags befindet sich ein `TextView`-Tag. Dieses definiert die gleichen Attribute `layout_height` und `layout_width` für die Größenanzeige des `TextView`-Widgets und ein weiteres Attribut `android:text` mit dem Inhalt: "Hello World, HalloApp".

Alle anderen Widgets von Android können durch ähnliche Layoutdefinitionen einer Oberfläche hinzugefügt werden, wie dies jedes Mal im Detail mit den nachfolgenden Aufgaben gezeigt wird.

Im Unterverzeichnis `values` von `res` finden Sie die Datei `strings.xml`, die in diesem Beispiel nur den Namen der App festlegt. Sie benutzt für die Definition eine String-Ressource, die noch in diesem Unterkapitel näher beschrieben wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
   <string name="app_name">HalloApp</string>
</resources>
```

Beim Bilden des Projekts wurde eine Java-Klasse mit dem Namen `HalloApp` implementiert, die die Start-Activity der App definiert und im Unterverzeichnis `src\jung\java8hallo` abgelegt ist:

```
package jung.j8hallo;
import android.app.Activity;
import android.os.Bundle;
public class HalloApp extends Activity {
   /* Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.main);
  }
}
```

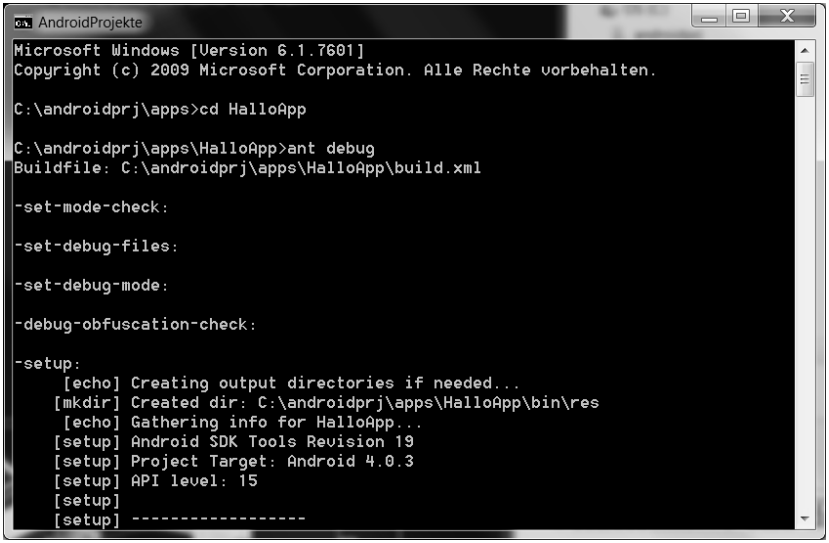
Sie ist von der Standard-Klasse `Activity` abgeleitet und überschreibt deren `onCreate()`-Methode. Darin wird die gleichnamige Methode ihrer Oberklasse aufgerufen und über die vorher beschriebene `main.xml`-Datei die Benutzeroberfläche für die App gesetzt. Die Layoutdatei wird über die ID `R.layout.main` als Ressource identifiziert. Dabei bezeichnet `R` den Namen einer Datei `R.java`, die vom `aapt`-Tool beim Bilden der App generiert wird.

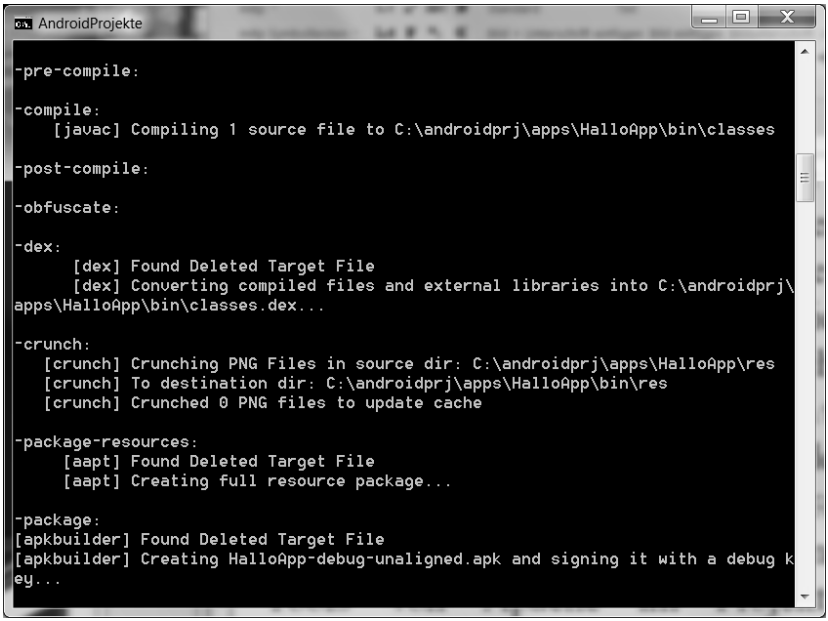
Um diesen Vorgang anzustoßen, wird das `ant`-Tool von Apache im Projektverzeichnis `C:\androidprj\apps\HalloApp` aufgerufen. Dieses führt die beim Anlegen des Projekts standardmäßig erstellte Datei zu diesem Zweck, `build.xml`, aus. Es kann im Debug- bzw. Release-Mode gestartet werden:

Mit `ant debug` wird eine App zum Testen und Debuggen gebildet, die resultierende `.apk`-Datei wird im `bin`-Verzeichnis des Projekts mit dem Namen `HalloApp-debug.apk` gespeichert.

Mit `ant release` wird eine App für die Freigabe an den User erstellt, die für die Benutzung signiert werden muss.

Wir rufen `ant debug` auf:





und sehen uns vor der App-Installation die neu erzeugten Ordner und den Inhalt der `R.java`-Datei genau an:

```
package jung.j8hallo;
public final class R {
   public static final class attr {
   }
   public static final class drawable {
      public static final int ic_launcher=0x7f020000;
   }
   public static final class id {
      public static final int textview=0x7f050000;
   }
   public static final class layout {
      public static final int main=0x7f030000;
   }
   public static final class string {
      public static final int app_name=0x7f040000;
   }
}
```

Diese Datei wird in einem neuen Unterverzeichnis `gen` von `HalloApp` hinterlegt und in ihr werden alle Ressourcen des Projekts, die im Unterverzeichnis `res` des

App-Verzeichnisses abgelegt sind, eingetragen. Sie bilden zusammen mit den Layoutdateien und Java-Klassen die drei wichtigsten Bausteine einer App.

Während der Aufbau von Java-Klassen und XML-Dateien (z.B aus den Java-Übungsbüchern) bekannt ist, wollen wir uns mit der Verwaltung und dem Zugriff auf Ressourcen bereits an dieser Stelle etwas näher auseinandersetzen. Wir beginnen mit denjenigen, die sich auch in der Datei `R.java` dieses Projekts befinden: Layout-, String- und Drawable-Ressourcen.

Wie der `R.java`-Datei zu entnehmen ist, werden allen Ressourcen eine eindeutige ID und ein Ressourcentyp zugeordnet. Für jeden dieser Typen fügt der Ressourcen-Compiler eine `static-final`-Klassendefinition ein, wie z.B.:

```
public static final class layout {
   public static final int main=0x7f030000;
}
```

Wenn auf diese Ressourcen im Java-Code zugegriffen wird, muss ihre ID mit dem Präfix `R.` gefolgt vom Ressourcentyp angegeben werden: `setContentView (R.layout.main)` bzw. `TextView androidText= (TextView)findViewById (R.id.textview)`.

Der Ressourcentyp wird über das Verzeichnis, in dem eine Ressource abgelegt ist, bestimmt. So werden im `layout`-Unterverzeichnis von `res` Layoutdefinitionen wie die obige `main.xml` erwartet und unter `res\values .xml`-Dateien, die ein `<resources>`-Tag beinhalten. Einem `<resources>`-Tag können wiederum unterschiedliche Elemente, darunter `<string>`, `<string-array>`, `<integer-array>`, `<color>`, `<array>` etc. untergeordnet werden.

Damit können z.B. String-Werte als String-Ressourcen in der Datei `strings.xml` hinterlegt werden:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">HalloApp</string>
   <string name="hallo">Hallo Android</string>
</resources>
```

Der Zugriff darauf erfolgt über den dabei vergebenen Namen mit `@string/hallo` in einer Layoutdatei bzw. `R.string.hallo` in Java-Klassen.

Um Bilder als Ressourcen zu verwalten, können diese im `res\drawable`-Ordner abgelegt werden. Dabei unterstützte Formate sind: `JPG`, `PNG`, `BMP` und `GIF`. Der Zugriff darauf kann aus einer `.xml`-Datei mit: `android:src="@drawable/android3"` oder aus einer Java-Klasse z.B. mit `androidImage.setBackground-Resource(R.drawable.android3)` erfolgen, wobei `android3` den Namen der Datei bezeichnet.

In vielen Fällen reicht jedoch allein die Vergabe einer ID für einen Zugriff auf die Inhalte von Ressourcen im Java-Code nicht aus. Das Paket `android.content.res` definiert mehrere Klassen für den Zugriff auf die Ressourcen von Applikationen. Eine dieser Klassen ist die Handler-Klasse `Resources`. Grundsätzlich kann zum Erwerben der einer Applikation zugeordneten `Resources`-Instanz die Methode `getResources()` der Klasse `android.content.Context`, die die Applikationsumgebung definiert, aufgerufen werden.

Unter `http://developer.android.com/guide/topics/resources/available-resources.html` finden Sie eine Aufstellung aller Ressourcentypen von Android.

So können beispielsweise in der Datei `strings.xml` oder in einer XML-Datei mit einem beliebigen Namen aus dem Verzeichnis `res\values` String-Arrays definiert werden,

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string-array name="buecher">
      <item>Java 7 Das Übungsbuch Band I</item>
      <item>Java 7 Das Übungsbuch Band II</item>
   </string-array>
</resources>
```

die hauptsächlich zum Speichern von Daten in `Spinner`- und `ListView`-Widgets eingesetzt werden.

Referenziert wird diese Ressource in XML mit `@[package:]array.buecher` und im Java-Code mit `R.array.buecher`: `Resources resource = getResource(); string[] buecher = resource.getStringArray(R.array.buecher)`.

Farben werden in Android als hexadezimale RGB(Alpha-Red-Green-Blue)-Werte in einem der Formate #RGB, #ARGB, #RRGGBB, #AARRGGBB angegeben, wie z.B. mit der Eigenschaft `android:background="ffff0000"` in einer Layoutdefinition.

Auch diese können als Color-Ressourcen definiert werden, indem sie in einer Datei mit einem beliebigen Namen, wie z.B. `colors.xml` gespeichert und mit einem Namen versehen werden:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="red">ffff0000</color>
   <color name="magenta">ff00ff00</color>
</resources>
```

Analog zu String-Ressourcen wird in XML-Attributen damit die obige Eigenschaft mit `android:textColor="@color/red"` festgelegt. Im Java-Code kann mit:

```
Resources resource = getResource(); int color = resource.getColor(R.color.red);
```

darauf zugegriffen werden.

Die Klasse `TypedArray` ist wie auch die Klasse `Resources` Bestandteil des Pakets `android.content.res`. Sie kann benutzt werden, um Arrays von anderen Ressourcen wie z.B. von Drawables zu erzeugen. Ein derartiges Array kann gleichzeitig mehrere Ressourcentypen enthalten und mit einem beliebigen Namen wie z.B. `arrays.xml` im Ordner `res\values` abgelegt werden:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <array name="icons">
      <item>@drawable/blume</item>
      <item>@drawable/hortensien</item>
   </array>
   <array name="colors">
      <item> ffff0000</item>
      <item> ff00ff00</item>
   </array>
</resources>
```

wobei `blume.jpg` und `hortensien.jpg` zwei Bilddateien aus dem Ordner `res\drawables` bezeichnen (siehe dazu die Aufgabe 2.3).

Referenziert werden so definierte Ressourcen in XML mit `@[package:]array.icons` bzw. `@[package:]array.colors`. Im Java-Code können die Ressourcen-IDs für den Zugriff auf Komponenten wie folgt benutzt werden:

```
Resources resource=getResource();
TypedArray icons=resource.obtainTypedArray(R.array.icons);
Drawable drawable=icons.getDrawable(0);
```

bzw.

```
TypedArray colors= resource.obtainTypedArray(R.array.colors);
int color =colors.getColor(0,0)
```

`android.graphics.drawable.Drawable` ist eine abstrakte Klasse. Wie der Dokumentation zu entnehmen ist, repräsentiert in Android ein Drawable eine Abstraktion für »alles, was gezeichnet werden kann« (siehe dazu auch die Aufgabe 3.3).

Ein erster Schritt in der Programmierung mit Java ist, zu verstehen, dass von abstrakten Klassen und Interfaces keine Instanzen gebildet werden können.

Eine abstrakte Klasse kann abstrakte Methoden enthalten, dies sind Methoden ohne Implementierungen. Generell ist eine abstrakte Klasse dazu da, um die gemeinsamen Eigenschaften und Funktionalitäten mehrerer Klassen in Form von Feld- und Methodendefinitionen zu sammeln und als deren Oberklasse eingesetzt

zu werden. Eine Unterklasse, die von einer abstrakten Klasse abgeleitet ist, kann nur dann instanziiert werden, wenn sie alle ihre abstrakten Methoden durch Überschreiben implementiert. Ansonsten muss sie auch als `abstract` deklariert werden. Nicht abstrakte Methoden einer abstrakten Oberklasse werden geerbt und können eventuell überschrieben werden. Es können jedoch Referenzen vom Typ einer abstrakten Klasse definiert werden, die auf Instanzen von konkreten Klassen, die diese erweitern, verweisen.

Wie Java-Klassen definieren auch Interfaces einen Referenztyp. In der Definition einer Schnittstelle werden die Schlüsselwörter `class` und `abstract` durch `interface` ersetzt.

Im Unterschied zu Klassen ist ein Interface eine reine Spezifikation. Interfaces definieren Verhaltensweisen, geben Methoden vor, ohne diese zu implementieren, und können parallel dazu nur Konstanten, die als `static` und `final` deklariert werden, enthalten. Dadurch, dass alle Methoden eines Interface implizit `abstract` und `public` sind, kann ein Interface weder Konstruktoren noch Klassenmethoden definieren.

Wie auch im Fall abstrakter Klassen können keine Objekte vom Typ eines Interface erzeugt werden. Es können Referenzen vom Typ eines Interface definiert werden und diese können auf beliebige Objekte von konkreten Klassen zeigen, die entweder die Schnittstelle implementieren oder von einer abstrakten Klasse, die diese Schnittstelle implementiert, abgeleitet wurden. An den so erzeugten Objekten können alle Methoden der Schnittstelle aufgerufen werden. Sind weitere Details gewünscht, können diese z.B. im Kapitel 3 von »Java 7 Das Übungsbuch Band I« Seite 145 gefunden werden.

Style-Ressourcen, die in Android Stile definieren, können mithilfe eines `<stiles>`-Elements innerhalb eines `<resources>`-Tags ebenfalls in `.xml`-Dateien des `values`-Ordners untergebracht werden. Sie spielen eine ähnliche Rolle wie CSS-Stylesheets in der HTML-Sprache für das Webdesign (dazu können Details im Übungsbuch Band III »Servlets und JavaServer Pages« Kapitel 2, Seiten 26 und 234 nachgeschlagen werden).

Style-Ressourcen definieren das Aussehen und das Format der UI-Elemente von App-Oberflächen. Damit kann das Design einer Oberfläche vom Inhalt getrennt werden. So kann beispielsweise für das in einer Layoutdatei `main.xml` definierte `TextView`-Element:

```
<TextView
   android:textSize">20sp</item>
   android:textColor">#008</item>
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Hello, World!" />
```

eine `styles.xml`-Datei (der Name kann ebenfalls beliebig gewählt werden) erstellt werden, die im Verzeichis `values` gespeichert wird und die Design-Eigenschaften des TextViews beinhaltet:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="customText">
     <item name="android:textSize">20sp</item>
     <item name="android:textColor">#008</item>
   </style>
</resources>
```

Um diese Style-Ressource dem TextView zuzuordnen, muss in der `main.xml`-Datei folgende Änderung vorgenommen werden:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
  style="@style/customText"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Hello, World!"/>
```

Das `parent`-Attribut eines `<style>`-Tags ist optional und spezifiziert die Ressourcen-ID eines anderen Styles, von dem dieser Style Eigenschaften erben kann.

Ein Theme ist ein Style, der für eine ganze App oder Activity gesetzt wird. Weitere Details dazu können unter: `http://developer.android.com/guide/topics/resources/style-resource.html` nachgeschlagen werden.

Mit den `openRawResource()`-Methoden der Klasse `Resources` kann ein Read Wrapper für das Lesen von sogenannten Raw-Ressourcen ermittelt werden. Damit sind die Dateien aus dem `res\drawable`- und `res\raw`-Verzeichnis einer Applikation gemeint. Die im `raw`-Ordner abgelegten Dateien (Bilder, Sounds, Videos oder auch Textdateien) verfügen ebenfalls über eine Ressourcen-ID für den Zugriff. In diesen Methoden können aber weder String- noch Color-Ressourcen übergeben werden.

Die Klasse `Resources` liefert gleichzeitig eine High-level-API für den Zugriff auf Dateien, die im Verzeichnis `assets` der App hinterlegt sind und über keine Ressourcen-ID verfügen. Dazu muss mit ihrer `getAssets()`-Methode ein AssetManager ermittelt werden. Zum Ermitteln einer `AssetManager`-Instanz kann auch direkt die Methode `getAssets()`, die die `Activity`-Klasse von der Klasse `Context` erbt, aufgerufen werden.

Die Klasse `AssetManager` wiederum liefert eine Low-level-API, die gebraucht wird, um die vom Benutzer im `assets`-Verzeichnis abgelegten Dateien mit einer App als einfache byteorientierte Streams zu verbinden, zu öffnen und zu lesen.

Weitere Details zum Öffnen und Lesen von Dateien aus den Verzeichnissen `res\raw` und `assets` können dem Unterkapitel 2.7 entnommen werden.

Wir werden alle hier beschriebenen Ressourcentypen in die nachfolgenden Aufgaben integrieren, um konkret zu zeigen, wie ein Zugriff darauf erfolgen kann.

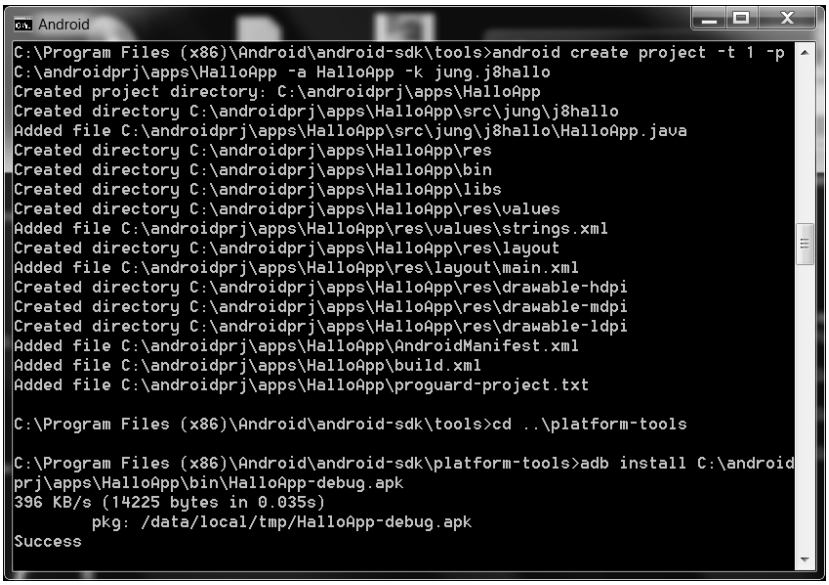
1.7 Die Installation von Android-Applikationen

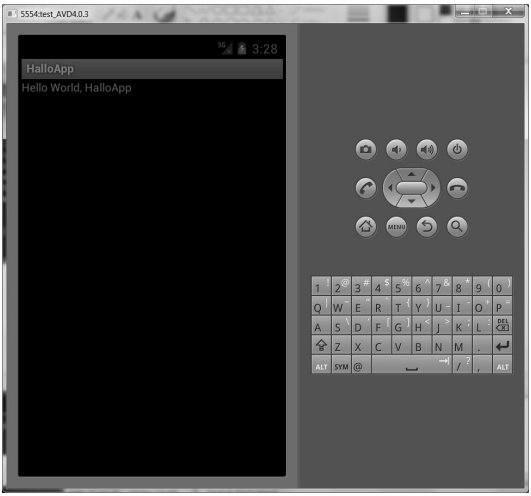
Bevor eine App für den Emulator installiert wird, muss das AVD-Tool gestartet werden und es muss abgewartet werden, bis dieses den vollständigen Android-Bildschirm anzeigt. (Wie eine App auf dem Smartphone installiert werden kann – unter Benutzung der gleichen APK-Datei – wird in Kapitel 3 gezeigt.)

Danach rufen wir im Verzeichnis `Android\android-sdk\platform-tools` das `adb`-Tool mit: `adb install C:\androidprj\apps\HalloApp\bin\HalloApp-debug.apk` auf.

Die Android Debug Bridge (adb) ist ein Client-Server-Programm, das eine Kommunikation mit dem Emulator oder einem Android-Gerät gewährleistet.

Nach korrekter Ausführung wird am Bildschirm des Emulators der Name `HalloApp` in der Liste der Applikationen eingetragen und diese kann mit einem Klick mit der linken Maustaste aufgerufen werden.





Aufgabe 1.1

☆

Die erste eigene App

Wie im Unterkapitel 1.6 beschrieben wurde, ist es in der App-Programmierung üblich, Text- und Bild-Komponenten als Ressourcen zu verwalten. Texte können in der Datei `strings.xml` im `values`-Unterverzeichnis von `res` hinterlegt werden und Images in einem Unterverzeichnis `drawable` von `res`. Ändern Sie die zuvor erstellte App dahin gehend ab, dass der angezeigte Text nicht direkt in das `text`-Attribut des TextViews eingetragen wird, sondern diesem ein Verweis auf eine Ressource, in der der Text gespeichert ist, aus der Datei `strings.xml` zugewiesen wird. Unsere erste eigene Android-App (mit dem gleichen Namen) soll den Text »Hallo Android« am Bildschirm ausgeben und ein von Ihnen ausgewähltes Android-Icon am Bildschirm anzeigen.

Erweitern Sie dazu die Datei `main.xml` um einen ImageView-Eintrag und ordnen Sie dem damit beschriebenen View-Element, wie auch dem TextView mit `android:id = "@+id/imageview"` bzw. `android:id = "@+id/textview"` eine Ressourcen-ID zu, damit diese Elemente im Java-Code identifiziert werden können. Eine derartige ID beginnt wie bereits erwähnt mit dem `@`-Zeichen gefolgt von dem Präfix `+id`. Somit können diese Ressourcen mit `R.id.imageview` bzw. `R.id.textview` in einer Java-Klasse angesprochen werden.

Erstellen Sie unter `res` ein neues Unterverzeichnis `drawable` und legen Sie eine von vielen Image-Dateien mit dem Android-Maskottchen in diesem Verzeichnis ab.

Fügen Sie der Datei `strings.xml` den Eintrag: `<string name="hallo">Hallo Android</string>` hinzu und nach einem Blick in die Android-Dokumentation oder in die »Hinweise für die Programmierung« zu dieser Aufgabe neue `android`-Attribute der Layoutdefinition, um Text und Image in einer anderen Farbe und Größe darzustellen. Die Bilddatei kann, wie bereits in der theoretischen Beschreibung erwähnt wurde, mit dem Attribut `android:src="@drawable/android3"`, wobei `android3.jpg` ihren Namen bezeichnet, dem ImageView zugeordnet werden.

Um ein besseres Verständnis für die Programmierung von Android-Apps allgemein und für das Zusammenspiel von XML- und Java-Klassen bzgl. der Definition von Oberflächen zu erlangen, soll diese App wie in Java üblich auch ohne Layoutdatei programmiert werden.

Definieren Sie dazu eine zweite Activity mit dem Namen `HalloAppohneLayoutDatei` in deren `onCreate()`-Methode alle View- und ViewGroup-Elemente über die Konstruktoren der Klassen wie folgt erzeugt werden:

```
   LinearLayout linearLayout = new LinearLayout(this);
// Die Größe und Ausrichtung des Layouts festlegen
   linearLayout.setLayoutParams(new LayoutParams(
     LayoutParams.FILL_PARENT,LayoutParams.FILL_PARENT));
   linearLayout.setOrientation(LinearLayout.VERTICAL);
   TextView textView = new TextView(this);
// Der TextView-Instanz einen Text zum Anzeigen hinzufügen
   textView.setText("Hallo Android");
   ImageView imageView = new ImageView(this);
```

Sobald ein Bild in einem der `res\drawable`-Verzeichnisse abgelegt wurde, ist der Name des Bildes als Ressourcen-ID verfügbar und die Bilddatei kann durch Angabe der Ressourcen-ID in das ImageView zur Laufzeit der App geladen werden:

```
   imageView.setImageResource(R.drawable.android3);
```

Als Alternative dazu kann die Bilddatei in eine `Bitmap`-Instanz (binäres Objekt) umgesetzt werden, die dem View-Element zugewiesen wird:

```
   Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
     R.drawable.android3); imageView.setImageBitmap(bitmap);
```

Um die Anzeige im ImageView auszurichten, kann die Höhe und Breite eines Bildes mit der Methode `scaleType()` angepasst werden. `FIT_XY` (z.B.) sagt aus, dass das Bild ungeachtet der Seitenverhältnisse im Container komplett angezeigt werden soll: `imageView.setScaleType(ImageView.ScaleType.FIT_XY)`.

Wird eine Layoutdatei benutzt, kann diese Eigenschaft mit `android:scaleType="FIT_XY"` für das ImageView gesetzt werden.

Hinweise für die Programmierung:

Zum Testen der zweiten Activity muss diese als Start-Activity in der `AndroidManifest.xml`-Datei eingetragen werden. Es kann immer nur eine Activity als Start-Activity benutzt werden. Sie können in diesem Fall für den Test die Datei `main.xml` aus dem `res\layout`-Verzeichnis löschen, der Compiler wird nicht nach einer Layoutdatei suchen, wenn kein Verweis auf diese in der Java-Klasse enthalten ist.

Wir stellen fest, dass der wesentliche Unterschied zwischen dem Benutzen bzw. Weglassen von XML-Code darin besteht, dass im ersten Fall die ViewGroup- und View-Elemente nicht explizit instanziiert werden müssen, da dies bereits durch deren Einfügen in die Layoutdatei mittels XML-Tags geschieht. Die Trennung der Darstellung einer Oberfläche (in der XML-Datei) von der Programmlogik (in Java-Klassen) macht ein Projekt, dessen Ergebnis eine App ist, übersichtlicher und ist auch der Standard in Android.

Farb- und Schrift-Eigenschaften für ein TextView können in der XML-Datei wie folgt gesetzt werden:

```
<TextView android:text="Hallo Android"
android:id="@+id/textview"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="20px"
android:textStyle="bold"
android:textColor="#FFFF0000">
```

Die Auflistung aller zugehörigen XML-Attribute sowie deren Zuordnung zu den verwandten Methoden finden Sie unter `http://developer.android.com/reference/android/widget/TextView.html`.

Java-Dateien: `HalloApp.java`, `HalloAppohneLayoutDefinition.java`
XML-Dateien: `AndroidManifest.xml`, `main.xml`, `strings.xml`

1.8 Lösungen

Lösung 1.1

Die Klasse HalloApp

```
package jung.j8hallo;
import android.app.Activity;
import android.os.Bundle;
```

```
import android.widget.ImageView;
import android.widget.TextView;
public class HalloApp extends Activity {
// Die onCreate()-Methode der Activity-Oberklasse überschreiben
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
// Anstatt Text und Bild direkt über Attributwerte in der
// main.xml-Layoutdatei zu setzen, können diese in der Ressourcen-
// Datei strings.xml zur Verfügung gestellt und über ihre
// Ressourcen-ID angesprochen werden; das heißt, dass eine Mischung
// von Zuweisungen mit XML und Java durchaus möglich ist
  /*  TextView androidText= (TextView)findViewById(
                                    R.id.textview);
      androidText.setText(R.string.hallo);
      ImageView androidImage = (ImageView)findViewById(
                                    R.id.imageview);
      androidImage.setBackgroundResource(R.drawable.android3);
  */
    }
}
```

Die Datei AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=
            "http://schemas.android.com/apk/res/android"
   package="jung.j8hallo"
   android:versionCode="1"
   android:versionName="1.0">
   <application android:label="@string/app_name"
            android:icon="@drawable/android8">
      <activity android:name="HalloApp"
            android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name=
                  "android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Die Datei main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
            "http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:id="@+id/textview"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:text="@string/hallo"
  />
  <ImageView
    android:id="@+id/imageview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:src="@drawable/android3"
  />
</LinearLayout>
```

Die Datei strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">HalloApp</string>
  <string name="hallo">Hallo Android</string>
</resources>
```

Die Klasse HalloAppohneLayoutDatei

```
package jung.j8hallo;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.LinearLayout;
import android.widget.LinearLayout.LayoutParams;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Color;
public class HalloAppohneLayoutDatei extends Activity {
// Globale Referenzen vom Typ der ViewGroup- und View-Elemente
```

```
   private TextView textView;
   private ImageView imageView;
   private LinearLayout linearLayout;
// Die onCreate()-Methode der Activity-Oberklasse überschreiben
@Override
   public void onCreate(Bundle savedInstanceState) {
// Die gleichnamige Methode der Oberklasse aufrufen
     super.onCreate(savedInstanceState);
// View-Elemente über einen Konstruktoraufruf instanziieren
     linearLayout = new LinearLayout(this);
     linearLayout.setLayoutParams(new LayoutParams(
     LayoutParams.FILL_PARENT,LayoutParams.FILL_PARENT));
     linearLayout.setOrientation(LinearLayout.VERTICAL);
     textView = new TextView(this);
     textView.setText("Hallo Android");
     textView.setTextColor(Color.parseColor("#00FF00"));
     imageView = new ImageView(this);
// Sobald ein Bild in einem der res/drawable-Verzeichisse
// abgelegt wurde, ist der Name des Bildes als Ressourcen-ID
// verfügbar; die Bilddatei durch Angabe der Ressourcen-ID in das
// ImageView zur Laufzeit der App laden
     imageView.setImageResource(R.drawable.android3);
// Anzeige im ImageView ausrichten; die Höhe und Breite eines
// Bildes kann über die Eigenschaft scaleType angepasst werden,
// die in der XML-Layoutdatei mit android:scaleType="FIT_XY"
// gesetzt werden kann; FIT_XY sagt aus, dass das Bild ungeachtet
// der Seitenverhältnisse im Container komplett angezeigt wird
     imageView.setScaleType(ImageView.ScaleType.FIT_XY);
// Als Alternative kann die Bilddatei in eine Bitmap-Instanz
// umgesetzt werden, die dem View-Element zugewiesen wird
     // Bitmap bitmap  = BitmapFactory.decodeResource(
     // getResources(), R.drawable.android3);
     // imageView.setImageBitmap(bitmap);
     linearLayout.addView(textView);
     linearLayout.addView(imageView);
     setContentView(linearLayout);
   }
}
```

Programmausgaben