Hochschule München

Fakultät 07 Informatik und Mathematik

Seminararbeit

Android GUI Entwicklung mit Native Java

Verfasser: Stefan Haupt

Matrikelnummer: 06479910

Fach: Aktuelle Technologien verteilter Java Anwendungen (JEE)

Eigentumserklärung:

Ich, Stefan Haupt, erkläre durch diese Unterschrift, dass ich diese Seminararbeit selbständig erstellt, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Unterschrift

Hochschule München
Fakultät 07 Informatik und Mathematik
Lothstrasse 64
D - 80335 München

Inhaltsverzeichnis

1 Einführung und Motivation 1
1.1 Mobile Applikationen 1
1.2 Android als Betriebssystem 1
1.3 Java als Programmiersprache 2
1.4 Entwicklungswerkzeuge 2
2 Android Applikationen 4
2.1 Activities 5
2.2 Ressourcen 7
3 Layouts 11
3.1 Layout Hierarchie 11
3.2 LinearLayout 11
3.3 RelativeLayout 14
3.4 ListView und GridView 15
4 UI-Komponenten 17
4.1 Standard Elemente 17
4.2 Sonstige Elemente 21
4.3 Benachrichtigungen 22
5 Input Events 23
5.1 EventListeners 23
5.2 Events 24
6 Kommunikation 25
6.1 Explizite Intents 25
6.2 Implizite Intents 25
7 Erste GUI entwickeln 26
7.1 Eclipse Projekt anlegen 26
7.2 Activity inklusive Layout erstellen 28
7.3 Komponenten einfügen 29
7.4 Handling 31
7.5 Alternative Ressourcen hinzufügen 36
8 Ausblick und Fazit 40
Literaturverzeichnis 41

Listing- und Abbildungsverzeichnis

Abbildung 1 – Fragmente für Layouts 7
Abbildung 2 – res-Ordner im Package Explorer von Eclipse 8
Abbildung 3 – Default Layout 9
Abbildung 4 – Alternative Layouts 9
Abbildung 5 – Layout Hierarchie 11
Abbildung 6 – LinearLayout-Beispiel 12
Abbildung 7 – RelativeLayout Beispiel 15
Abbildung 8 – Anordnung von Items in einem ListView 16
Abbildung 9 – Anordnung von Items in einem GridView 16
Abbildung 10 – Spinner als Menü 20
Abbildung 11 – TimePicker und DatePicker 20
Abbildung 12 – Dialog für Helligkeitseinstellungen 21
Abbildung 13 – Android Application - Project Wizard 26
Abbildung 14 – Package Explorer 27
Abbildung 15 – Unsere Test-App 31
Abbildung 16 – Notification unserer TestApp 35
Abbildung 17 – Zweite Activity unserer TestApp 36
Abbildung 18 – Layout Hierarchie in der Outline-View 38
Abbildung 19 – Deutsche Lokalisation der Beispiel-App 38
Abbildung 20 – Ein Android 40

Listing 1 – Manifest-Datei 4
Listing 2 – onCreate Methode 5
Listing 3 – Layout-Beispiel 6
Listing 4 – LinearLayout-Beispiel 13
Listing 5 – RelativeLayout Beispiel 14
Listing 6 – Einfügen von Elementen in ein Layout 17
Listing 7 – TextView 17
Listing 8 – Radiobuttons in einer Radio Group 19
Listing 9 – Menu Beispiel 22
Listing 10 – Notification-Builder 22
Listing 11 – onClick-Event 23
Listing 12 – OnClickListener 23
Listing 13 – Expliziter Intent 25
Listing 14 – Impliziter Intent 25
Listing 15 – Angelegte Manifest-Datei 28

Listing 16 – Activity-Klasse wurde angelegt....28
Listing 17 – Activity in AndroidManifest.xml festgelegt....28
Listing 18 – LinearLayout für unsere MainActivity Klasse....29
Listing 19 – Komponenten im LinearLayout....30
Listing 20 – onClick-Handling für die Radiobuttons....32
Listing 21 – Buttons initialisiert....32
Listing 22 – ButtonCancelListener-Klasse....33
Listing 23 – ButtonOkListener-Klasse....33
Listing 24 – startTextviewChange-Methode....34
Listing 25 – Initialisierung von Notification-Builder und NotificationManager....34
Listing 26 – startNotification-Methode....34
Listing 27 – startSecondActivity-Methode....35
Listing 28 – Landscape-Layout....37

1 Einführung und Motivation

Dieses Dokument soll einen Überblick über die Möglichkeiten beim Entwickeln Mobiler Android-Anwendungen verschaffen. Ziel ist es die Grundkonzepte zu vermitteln, sodass nach dem Lesen dieses Dokuments bereits erste einfache Android GUIs (Graphical User Interfaces) entwickelt werden können. In diesem Dokument wird speziell auf die Entwicklung von Anwenderoberflächen eingegangen, welche als Schnittstelle für den End-Anwender dienen.

1.1 Mobile Applikationen

Mit dem stetig wachsenden Markt für Mobile Devices, steigen auch die Anforderungen an Applikationen für Mobil-Geräte. Die immer leistungsfähigere Hardware bietet Entwicklern die Möglichkeit immer komplexere Software für immer kleiner werdende Geräte zu schreiben.

Bei Mobilen Applikationen muss grundsätzlich zwischen zwei Typen unterschieden werden. Zum einen Web-Applikationen, die über den Web-Browser genutzt werden und auf der anderen Seite die klassischen Anwendungen, welche auf dem System installiert und ausgeführt werden. Ein weiteres Merkmal von diesen „Smart-Devices“ ist die Internetfähigkeit und die Tatsache, dass man mit dem Gerät permanent online sein kann.

Als solche sich sind mobile Applikationen keine Neuerfindung. Die neumodischen Smart Devices haben jedoch einen neuen Markt für die so genannten Apps geschaffen.

1.2 Android als Betriebssystem

Das Android Betriebssystem basiert auf einem multi-user Linux-Kernel, in dem jede Applikation als eigener „User“ gesehen wird. Aus Sicherheitsgründen weist das Betriebssystem jeder Applikation eine eindeutige ID zu. Diese ID ist nur dem Betriebssystem bekannt und somit nicht von den Applikationen selbst einsehbar. Zu diesem Sicherheitsfeature kommt noch hinzu, dass jede Applikation in einer eigenen Virtuellen Maschine und damit in einem eigenen System-Prozess läuft. Der Prozess kann bei Bedarf vom Betriebssystem beendet werden. Das ist zum Beispiel der Fall, wenn eine im Vordergrund laufende Applikation mehr Speicher benötigen sollte und deshalb Hintergrundaktivitäten beendet werden. Durch die genannten Features erfüllt das Android-Betriebssystem die Spezifikation des „principle of least privilege“[1]. Diese Spezifikation besagt, dass jedes Modul nur dann einen Zugriff auf Informationen und Ressourcen erhalten darf, wenn diese auch tatsächlich benötigt werden. Ein weiterer Vorteil, der für das Entwickeln von Apps für Android spricht, ist die quelloffene Struktur: Bis auf

[1] Vlg. Application Fundamentals – Android Developers Guides

Einzelne sicherheitskritische Komponenten (z.B. der „Play Store“) wurde Android unter der Apache OpenSource Lizenz von Google veröffentlicht (Apache Licence 2.0).

1.3 Java als Programmiersprache

Android Applikationen sind in der Programmiersprache Java geschrieben. Die Einzelnen Applikationen laufen dementsprechend in einer virtuellen Maschine (VM) namens „Dalvik“, welche speziell für mobile Applikation angepasst wurde. Dabei ist die Funktionsweise der Dalvik VM vergleichbar mit der der Java VM.

Da Apps, die für Android entwickelt werden, Java-Applikationen sind, ist es hilfreich bereits einige Kenntnisse mit der Programmiersprache Java zu besitzen.

1.4 Entwicklungswerkzeuge

Google bietet für Android auf der Website android.com eine vollständige Entwicklungsumgebung zum Download an. Diese Entwicklungswerkzeuge nennen sich allgemein Android Development Tools (ADT) und beinhalten neben einer erweiterten Entwicklungsumgebung (Eclipse), einen Android SDK Manager und eine Virtualisierungssoftware namens AVD-Manger (Android Virtual Devices).

1.4.1 Eclipse mit Android Plugins

Eclipse ist bereits ein bekanntes und beliebtes Entwicklertool. Vor allem weil es sich beliebig und je nach Bedarf durch den Benutzer erweitern lässt und es unter einer OpenSource Lizenz von der Eclipse Community veröffentlicht wurde. In dem Download Packet auf der Android-Website ist eine Eclipse-Version enthalten mit installiertem ADT-Plugin-Packet.

1.4.2 SDK Manager

Mit dem SDK Manager lassen sich unter anderem die unterschiedlichen API-Levels zu den Android-Versionen herunterladen. Zudem lassen sich sämtliche benötigten Tools installieren, zu denen auch die Plattform-Tools gehören. Die Plattform-Tools dienen dazu die Kommunikation zwischen der Entwickler-Maschine und dem Gerät über USB herzustellen, mit Hilfe der ADB-Schnittstelle (Android Debug Bridge). Ein weiteres dieser Tools ist der AVD Manager.

1.4.3 Android Virtual Devices Manager

Mit dem Android Virtual Devices Manager (AVD Manager) lassen sich mit Hilfe einer Graphischen Benutzeroberfläche virtuelle Android Geräte anlegen. Ein solches Virtuelles Android Gerät läuft in einer eigenen Virtuellen Maschine und wird über das ADB-Interface angesprochen. Gestartet werden kann der AVD Manager anhand des Menüs von Eclipse, dem SDK Manager oder über den Kommandozeilen-Befehl „android avd“ im Ordner <sdk-dir>/tools/.

Für ein virtuelles Android Gerät bietet der AVD-Manager die Möglichkeit diverse Hardware Features mit dem AVD-Manager zu konfigurieren: Die entsprechende Bildschirmgröße, eine Camera, eine physikalische Tastatur, wie viel Arbeitsspeicher das Gerät besitzen soll sowie die Prozessor-Architektur des Geräts kann konfiguriert werden.

Der AVD Manager stellt also eine gute Möglichkeit dar, die Kompatibilität unserer Applikation zu testen, da die End-Nutzer viele verschiedene Android Geräte und Versionen verwenden.

2 Android Applikationen

Grundlegender Bestandteil einer Android Applikation ist die Manifest-Datei (AndroidManifest.xml). Dabei handelt es sich um eine Konfigurationsdatei, welche die Komponenten einer Android-Applikation definiert. Das Betriebssystem erkennt die Komponenten einer Applikation indem es die Manifest-Datei ausliest. Zusätzlich werden in der Manifest-Datei unter anderen die folgenden Punkte festgelegt:

- Berechtigungen die eine App benötigt (so genannte Permissions), das kann das Auslesen von Kontaktdaten, Zugriff auf das Dateisystem oder Internet-Vollzugriff sein
- Der minimale und maximale SDK-Level, um festzustellen für welche Versionen des Betriebssystems die App geeignet ist. Beispielsweise: „Android API Level 17“ (Android 4.2)
- Hardware Features die für das Nutzen der Applikation vorausgesetzt werden, Kamera, Bluetooth oder ob das Touchscreen des Gerätes Multi-Touch-Gesten beherrscht
- Software-Bibliotheken die, abweichend von dem Android Framework, verwendet werden
- etc.

Nachfolgendes Listing zeigt eine solche Manifest-Datei.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ghazxul.contact"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-sdk
        android:minSdkVersion="17"
        android:targetSdkVersion="17" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.ghazxul.contact.MainActivity"
            android:label="@string/app_name" >
            ...
        </activity>
        ...
    </application>
</manifest>
```

Listing 1 – Manifest-Datei

In dieser Manifest-Datei wird innerhalb des <manifest>-Tags die Version der Applikation, die Berechtigungen und das SDK-Level Festgelegt. Zudem legt das Tag <applikation> diverse

applikations-spezifische Eigenschaften und Attribute fest. Wir sehen hier das <activity>-Tag, dieses legt eine Komponente fest, die dazu dient Benutzer-Interaktionen zu tätigen.

An dieser Stelle können auch weitere Bestandteile einer Applikation festgelegt werden. Es können weitere Aktivities mit dem <activity>-Tag eingefügt werden. Mit einem <service>-Tag lassen sich die Services definieren, sie sind vergleichbar mit den Diensten des Windows-Betriebssystems. Zudem gibt es die Tags <receiver> und <provider> für Broadcast Receiver bzw. Content Provider. Komponenten die nicht in der Manifest-Datei definiert wurden, werden vom System nicht erkannt und können somit auch nicht ausgeführt werden.

Im Rahmen dieses Dokuments werden ausschließlich Activities und die zugehörigen Komponenten erklärt und anhand einiger Beispiele auch angewandt.

2.1 Activities

Wie bereits erwähnt, dient eine Activity dazu Benutzer-Interaktionen zu ermöglichen. Eine Activity ist also vergleichbar mit einem Fenster in dem das User-Interface gezeichnet wird. Dieses Fenster füllt in der Regel den Bildschirm des Gerätes aus. Üblicherweise besteht eine Applikation aus mehreren Activities. Davon wird eine Activity als so genannte „Main“ Activity definiert, die beim Start der Anwendung angezeigt wird.

Eine Activity besteht aus zwei Teilen, zum einen die Layout-Datei im XML-Format und zum anderen die zugehörige Activity-Klasse, die die abstrakte Klasse „Activity“ erweitert. Es ist üblich, dass die beiden Dateien einer Namenskonvention folgen, z.B. activity_main.xml und MainActivity.java. Die von uns implementierte Activity-Klasse – MainActivity – wird in der Manifest-Datei (siehe Listing 1) als Activity deklariert.

2.1.1 Activity Klasse

Die wichtigste Methode, die von einer Activity-Klasse implementiert werden muss, ist „onCreate“. Die Methode onCreate wird vom System aufgerufen, um die Activity zu zeichnen. Mit der Methode „setContentView“ kann das Layout der Activity festgelegt werden. Dies kann zwar innerhalb des Quelltextes programmatisch gelöst werden, es ist jedoch komfortabler das Layout in einer Layout-Datei umzusetzen.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);
	...
}
```

Listing 2 – onCreate Methode

In Listing 2 sehen wir eine Implementierung der onCreate-Methode, welche bislang noch nicht viel Funktionalität beinhaltet, abgesehen davon, dass wir das Layout für unsere Activity festgelegt haben.

2.1.2 Layout-Datei

Die zu einer Activity gehörende Layout-Datei wird mit Hilfe der Markup-Sprache XML (Extentensible Markup Language) geschrieben. Eine Layout-Datei beschreibt den Aufbau einer Activity.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />

</LinearLayout>
```

Listing 3 – Layout-Beispiel

Listing 3 zeigt ein Beispiel für eine Layout Datei. Es wird hier nur eine GUI mit einem so genannten TextView gezeichnet, welches den Text „Hello World“ statisch anzeigt. Dass der Text „Hello World“ fest in der Layout-Datei geschrieben ist, sollte vermieden werden. Es werden hierfür so genannte Ressourcen definiert, siehe Kapitel 2.2.

2.1.3 Fragmente

Ein weiterer Bestandteil der Android GUI sind so genannte Fragmente. Fragmente sind erst seit Version 3.0 Bestandteil der Android API. Sie wurden eingeführt um Layouts für unterschiedliche Bildschirmgrößen sowie Hoch- und Querformat einfach gestalten zu können. Ein Fragment kann beliebig ein anderes Fragment ersetzen. Dies wird gerne genutzt um Navigation zu realisieren, ohne die Activity auszutauschen. Fragmente sind somit Bestandteile von Activities. Der Vorteil Fragmente zu nutzen liegt zudem darin, dass eine Activity nicht immer neu gezeichnet werden muss, z.B. bei einem Orientierungswechsel. Außerdem können einzelne Fragmente bei verschiedenen Bildschirmgrößen auf eine oder verschiedene Activities verteilt werden, siehe Abbildung 1.

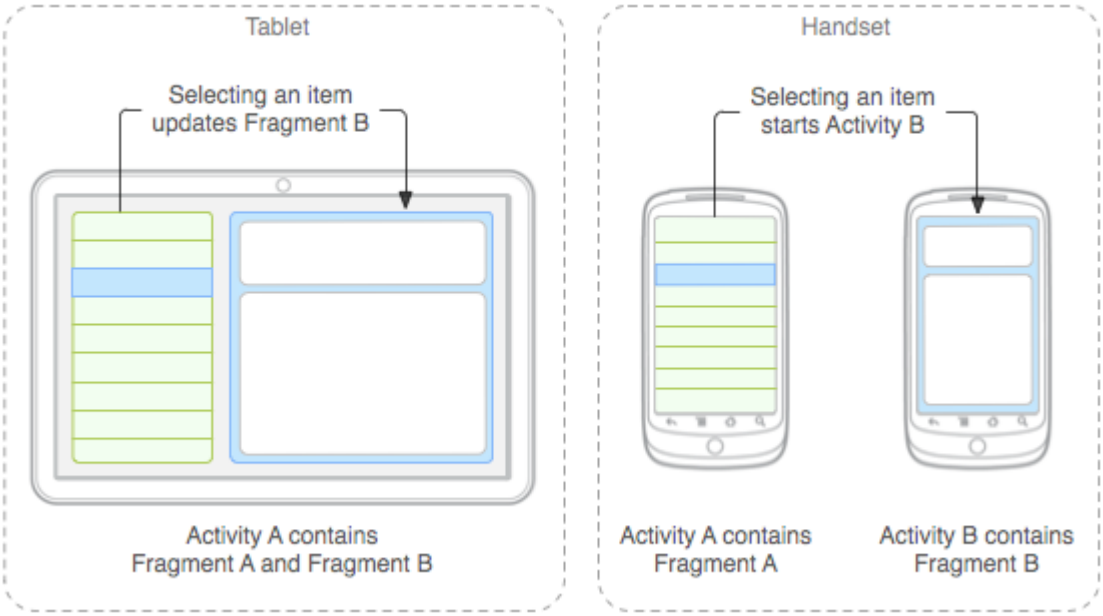

Abbildung 1 – Fragmente für Layouts[2]

Ein Fragment hat, ebenso wie eine Activity, eine zugehörige Klasse, die die abstrakte Klasse Fragment erweitert. Die Funktionsweise ist allgemein vergleichbar mit der einer Activity.

2.2 Ressourcen

Die Ressourcen einer Android-Applikation, zu dem auch das Layout gehört, werden im Projekt unter dem Ordner „res“ gespeichert. Die Ordner-Struktur ist dabei fest vorgegeben, da die Referenzen auf die einzelnen Komponenten automatisch vom SDK generiert und in der Klasse R abgelegt werden. Diese Klasse befindet sich im Source-Folder „gen“ und liegt im gleichen Package wie unsere Klassen. So können wir beim Schreiben des Quellcodes über die Klasse R auf unsere Komponenten zugreifen (z.B. mit `R.id.textview` oder `R.layout.layout_main`).

[2] Vgl. Fragments – Android Developers Guides

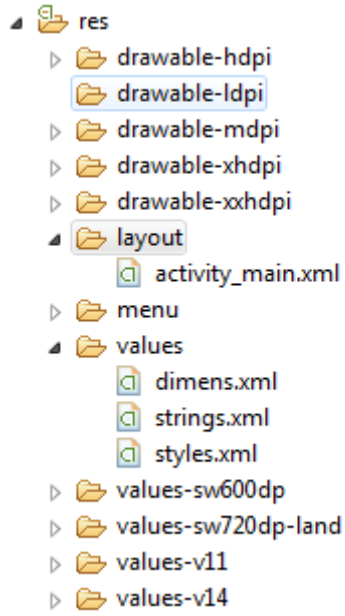

Abbildung 2 – res-Ordner im Package Explorer von Eclipse

In Abbildung 2 wird ein beispielhafter Aufbau eines solchen Res Ordners gezeigt. Die drawable-Ordner beinhalten die Bild-Dateien die gezeichnet werden sollen, abhängig von der Display-Größe des Geräts. Dazu gehört unter anderem das Icon der Applikation, das im Launcher angezeigt wird. In den values-Ordnern werden festgeschriebene, Werte global festgelegt, z.B. die Dimensionen (dimens.xml) oder die Strings (strings.xml) die in der GUI angezeigt werden sollen.

2.2.1 Alternative Ressourcen

Bei den Ressourcen gilt zu erwähnen, dass auch alternative Ressourcen definiert werden können, die abweichend von einer Default Ressource verwendet werden. Alternative Ressourcen können dazu dienen, dass abhängig von den gegebenen Eigenschaften eine andere Ressource verwendet wird.

Abbildung 3 – Default Layout[3]

Es ist manchmal nicht erwünscht, dass auf verschiedenen Geräten das gleiche Layout angezeigt wird. Wie in Abbildung 3 zu sehen, wird, für den Fall, dass nur ein Layout definiert ist, das Layout auch auf einem anderen Gerät (im Querformat) genauso angezeigt. Das ist häufig nicht schön und kann durch alternative Layouts vermieden werden. Diese können Layouts sein, die abhängig von der Bildschirmgröße und abhängig von der Orientierung (Hochformat/Querformat) gezeichnet werden, siehe Abbildung 4.

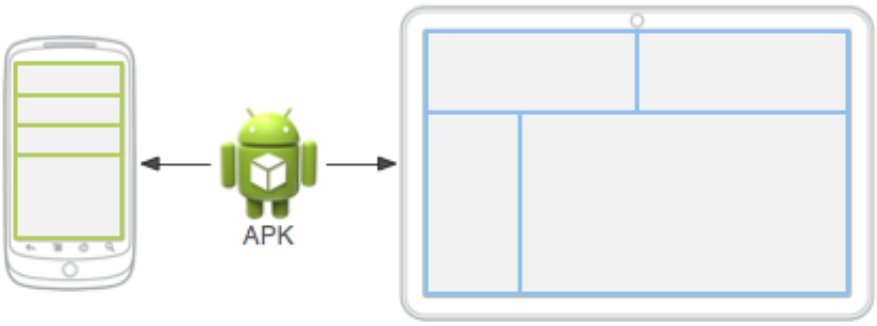

Abbildung 4 – Alternative Layouts[4]

Dieses Vorgehen hat den Vorteil, dass für Geräte mit anderer Displaygröße der Inhalt unterschiedlich dargestellt werden kann oder sogar zusätzliche GUI-Elemente im Querformat eingebaut werden können, die im Hochformat zu viel Platz einnehmen würden.

Die alternativen Ressourcen werden mit so genannten Qualifiern im Ordnernamen definiert.

Möchte man Beispielsweise zusätzliche Lokalisationen für unterschiedliche Gerätesprachen einfügen, kann man das dadurch realisieren, dass weitere values-Ordner mit einem entsprechenden Qualifier einer Sprache eingefügt werden.

---

[3] Vgl. Resources Overview – Android Developers Guides

[4] Vgl. Resources Overview – Android Developers Guides

Die Inhalte values-Ordner könnten dann demnach wie folgt aussehen.

- res/values/strings.xml, dies sind die default Strings, die in englische Sprache definiert werden sollten.

```
<resources>
	<string name="app_name">Hello World</string>
	<string name="text1">Hello!</string>
	...

</resources>
```

- res/values-de/strings.xml, der Qualifier „de“ steht für die deutsche Sprache

```
<resources>
	<string name="app_name">Hallo Welt</string>
	<string name="text1">Hallo!</string>
	...

</resources>
```

- res/values-ja/strings.xml, der Qualifier „ja“ für japanische Gerätesprache. Diese strings.xml wurde ohne den String für app_name definiert.

```
<resources>
	<string name="text1">Konnichiwa!</string>
	...
</resources>
```

Wird nun in der Applikation auf den String mit der ID app_name zugegriffen, wird zunächst geprüft, ob eine Variable mit der ID app_name in der strings.xml für die im System konfigurierte Sprache vorhanden ist. Sollte dies nicht der Fall sein, wird auf den Default-Wert zurückgegriffen. Im Falle der japanischen Sprache würde also der default Wert aus res/values/strings.xml „Hello World“ gezeigt und für die deutsche Sprachausgabe „Hallo Welt“.

3 Layouts

Da wir bereits einführend über Layouts gesprochen haben, sollte nun geklärt werden, wie Komponenten mit Hilfe verschiedener Layouts angeordnet werden können. Jede Komponente in einer Activity benötigt ein zugrunde liegendes Layout. Dabei ist zu beachten, dass die verschiedenen Layouts auch mit einander kombiniert werden können. Nachfolgend wird auf die wichtigsten Layouts eingegangen, da es im Rahmen dieses Dokuments zu tiefgreifend wäre, auf sämtliche Layout-Möglichkeiten einzugehen.

3.1 Layout Hierarchie

Ein Layout hat eine Hierarchie von weiteren Layouts und GUI Komponenten, die dem Layout untergliedert sind. Diese Hierarchie ergibt sich aus dem Aufbau der Layout-XML-Datei.

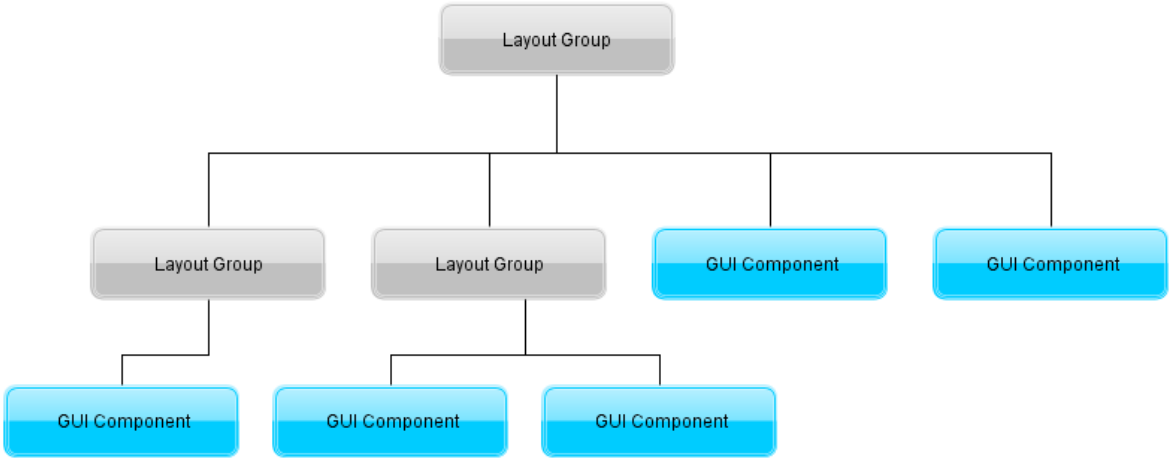

Abbildung 5 – Layout Hierarchie

Manche Layouts haben eine einfachere Hierarchie als andere und sind deshalb ein wenig ressourcenschonender. Dies ist zwar, wenn man die Geräte heutzutage betrachtet, nicht mehr sehr relevant. Dennoch sollte man den die Layout Hierarchie nicht aus den Augen lassen. Man sollte sich immer überlegen, ob die Hierarchie der GUI-Komponenten optimiert werden kann, ohne das eigentliche Layout zu ändern. Das Einsetzen von Fragmenten kann hier eine zusätzliche Rolle spielen.

3.2 LinearLayout

Bei dem LinearLayout handelt es sich um ein Layout bei dem sämtliche Kind-Elemente des Layouts nach einander in einer Reihe angeordnet werden. Dabei muss bestimmt werden, wie die Ausrichtung des Stapels sein soll: Horizontal oder Vertikal. Bei einer horizontalen Anordnung werden alle Elemente in einer Zeile angezeigt. Beim vertikalen Anordnen stehen die Elemente in einer Spalte.

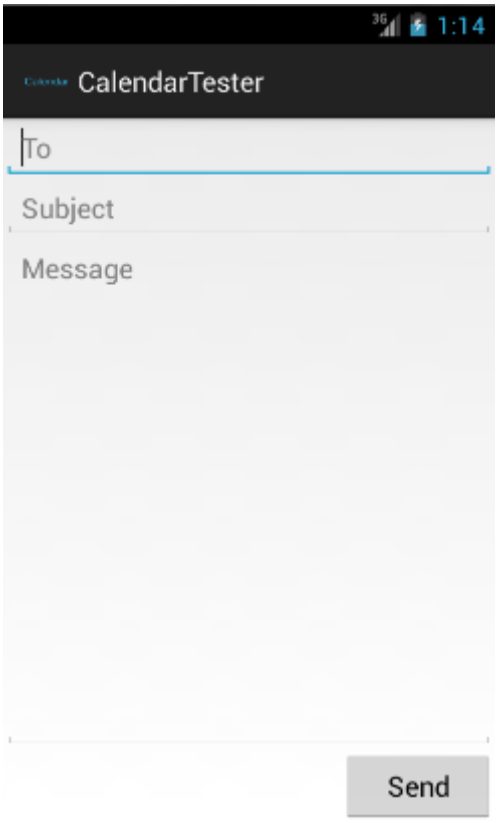

Abbildung 6 – LinearLayout-Beispiel

Ein LinearLayout könnte zum Beispiel in einer simplen Ansicht zum Schreiben einer E-Mail Verwendung finden, siehe Abbildung 6.

In einem Komponenten Tag lassen sich ebenso Breite und Höhe als Attribute angeben, sowie die Gravitation (rechts, zentral oder links) und die Gewichtung zur Priorisierung der Höhe und Breite. Die im Beispiel angewendeten Layout Einstellungen sind in Listing 4 zu finden.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="To" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Subject" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="Message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Send" />

</LinearLayout>
```

Listing 4 – LinearLayout-Beispiel

Die in Listing 4 zu sehenden Attribute android:layout_width und android:layout_height bestimmen die Breite und Höhe der Elemente. Die gewählten Optionen „wrap_content“ und „fill_parent“ bedeuten, dass die Größe an den Inhalt des Elements angepasst wird bzw. im Fall von „fill_parent“, dass die Komponente an die Größe des darüber liegenden Elements angepasst wird. Zudem können hier auch feste Pixelwerte angegeben werden, unabhängig vom Inhalt oder der Parent-Komponente.

Das LinearLayout ist aufgrund seiner Eigenschaften dafür geeignet Gruppen für eine Layout Anordnung zu erstellen. Diese Gruppen können dann wieder weitere Layouts sein.

3.3 RelativeLayout

Das RelativeLayout ermöglicht es, dass die Komponenten in Relation zu einander oder in Relation zur Parent-Komponente – also dem RelativeLayout – gezeichnet werden.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >

    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="Done" />
</RelativeLayout>
```

Listing 5 – RelativeLayout Beispiel

Aus dem RelativeLayout in Listing 5 ist zu sehen wie die einzelnen Komponenten gezeichnet werden. Die Entsprechende View zu dem obigen RelativeLayout ist in Abbildung 7 zu sehen.

Abbildung 7 – RelativeLayout Beispiel

Der große Vorteil des RelativeLayouts ist, dass die Layout Hierarchie viel schlanker ist. Es ist zwar theoretisch möglich, das Layout als LinearLayout zu realisieren, jedoch ist dann die Anzahl der einzelnen Komponenten bereits nahezu doppelt so hoch wie mit dem RelativeLayout.

Nachteil des RelativeLayouts ist, dass es bereits nach wenigen Komponenten recht schnell kompliziert und unübersichtlich werden kann und zudem muss man die sehr vielen Möglichkeiten der Anordnung zunächst kennen.

3.4 ListView und GridView

Eine ListView ist eine GUI-Komponente, welche eine scrollbare Liste darstellt. Die einzelnen Items werden mit Hilfe eines Adapters automatisch in die Liste eingefügt. Der Adapter kann sich dabei unterschiedlichster Quellen bedienen. Der Unterschied zu den bisherigen Layouts ist, dass eine ListView programmatisch mit Hilfe einer Adapter-Klasse gefüllt wird.

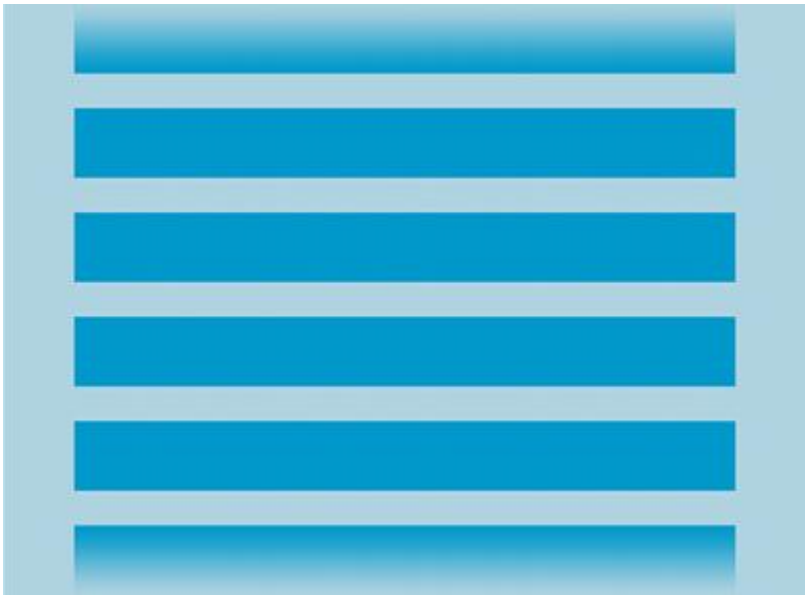

Abbildung 8 – Anordnung von Items in einem ListView[5]

Bei einer GridView handelt es sich um dasselbe Prinzip wie bei einer ListView mit dem Unterschied, dass die einzelnen Items in ein Raster eingeordnet werden müssen.

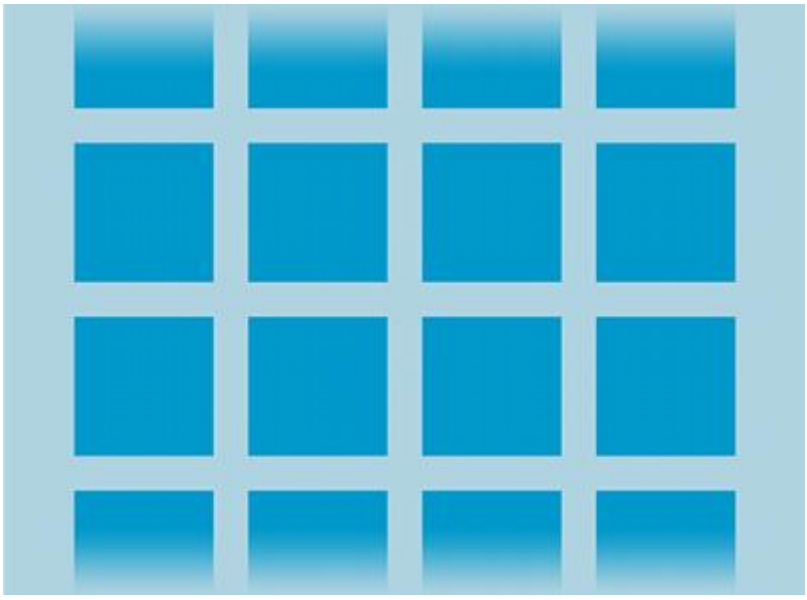

Abbildung 9 – Anordnung von Items in einem GridView[6]

[5] Vlg. ListView – Android Developers Guides

[6] Vlg. GridView – Android Developers Guides

4 UI-Komponenten

In diesem Kapitel werden die typischen Android GUI-Elemente und Techniken, wie sie auch in anderen Anwendungen Platz finden, beschrieben und ihre Verwendung erklärt. Komponenten werden durch das Einfügen in ein Layout einer Benutzeroberfläche hinzugefügt.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="16dp" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Hello World!" />

</RelativeLayout>
```

Listing 6 – Einfügen von Elementen in ein Layout

4.1 Standard Elemente

Die am häufigsten genutzten GUI-Elemente werden in den folgenden Unterkapiteln beschrieben.

4.1.1 Textview

Die Komponente Textview wurde bereits gezeigt. Es handelt sich dabei um die einfachste Komponente einer Android-GUI. Sie ist vergleichbar mit einem Label, das die Möglichkeit bietet einen Text anzuzeigen.

```
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Hallo Welt!" />
```

Listing 7 – TextView

4.1.2 Textfelder

Das EditText Element ist vergleichbar mit einem Textfeld, das dazu dient Text-Eingaben des Benutzers aufzunehmen. Das klingt relativ einfach, jedoch ist es immer problematisch Eingaben von Benutzern zu verarbeiten. Die Eingaben müssen auf alle Eventualitäten geprüft werden, was sich unter Umständen als schwierig erweisen kann.

Das EditText-Element bietet dem Entwickler Möglichkeiten die Kommunikation zwischen Applikation und End-Nutzer zu erleichtern.

- Mit einem **Hint** kann ein Feld mit einem Text vorbelegt werden als Hinweis, welche Art von Eingabe hier erfolgen soll. Sobald dann der Benutzer eine Eingabe tätigt verschwindet dieser Text.
- Den **Input Type** zu setzen hat den Vorteil, dass dem Benutzer eine Entsprechende Tastatur angezeigt wird, abhängig vom benötigten Input. Hier einige Beispiele:
    - `text` zeigt die normale Tastatur an
    - `textEmailAddress` zeigt ebenfalls die normale Tastatur an, mit zusätzlichen @-Zeichen
    - `number` zeigt die numerische Tastatur an
    - `textCapWords` jedes eingegebene Wort soll mit einem Großbuchstaben beginnen, für Titel, Vor- und Nachnamen
    - `textPassword` zeigt eine normale Tastatur mit dem Unterschied, dass jedes Zeichen im Textfeld durch einen Punkt ersetzt.

Somit hat der Entwickler einige Möglichkeiten, die Textfelder anzupassen und für die Benutzung zu optimieren.

4.1.3 Buttons

Ein Button-Element beinhaltet in der Regel ein Icon oder einen Text der dazu dient, die Aktion zu beschreiben die ausgeführt wird, sobald ein Button gedrückt wurde.

Zum einen gibt es den Standard-Button, der nur einen Text beinhaltet und zum anderen gibt es einen ImageButton, welcher in der Benutzeroberfläche nur ein Symbol anzeigt.

Events können auf zwei Arten einem Button zugewiesen werden:

- Attribut → Dem Button wird das Attribut android:onClick zugewiesen, die Eigenschaft des Attributs benennt dann die auszuführende Methode in der Activity oder dem Fragment.
- Listener → Dem Button kann programmatisch in der Activity (oder dem Fragment) ein OnClickListener angefügt werden, mit der Methode setOnClickListener.

Zusätzlich lassen sich Buttons in ihrer Erscheinung beeinflussen. Es ist möglich einen Button ohne Rand (BorderlessButtonStyle) oder einen benutzerdefinierten Hintergrund für den Button anzulegen.

4.1.4 Checkboxes

Checkboxes dienen in einer GUI dazu gewisse Zustände vom Benutzer anpassen zu lassen. Der Benutzer hat die Möglichkeit eine Checkbox anzuklicken um sie damit zu aktivieren bzw. zu

deaktivieren. Häufig finden Checkboxes Verwendung in Einstellungsansichten oder Auswahlansichten, bei denen der Benutzer Items in einer GUI an und abwählen kann.

Programmatisch kann der Zustand einer Checkbox mit den Methoden setChecked(boolean) oder toggle() umgestellt und mit isChecked() überprüft werden werden.

4.1.5 Radiobuttons

Radiobuttons sind, ähnlich wie Checkboxes, dafür da den Benutzer eine Auswahl treffen zu lassen. In der Regel werden Radiobuttons in einer zusammengehörenden Gruppe angezeigt. Innerhalb einer Gruppe kann immer nur ein Radiobutton aktiviert sein. Eine Solche Gruppe wird im Layout als RadioGroup definiert, die als Kind-Elemente die jeweiligen Radiobuttons beinhaltet.

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >

    <RadioButton
        android:id="@+id/rb_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onRadioButtonClicked"
        android:text="Hello World" />

    <RadioButton
        android:id="@+id/rb_user"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onRadioButtonClicked"
        android:text="Dear User!" />
</RadioGroup>
```

Listing 8 – Radiobuttons in einer Radio Group

4.1.6 Togglebuttons

Ein Togglebutton hat prinzipiell die gleiche Funktionalität wie eine Checkbox. Es wird vom Benutzer entschieden ob eine gewisse Funktionalität aktiviert oder deaktiviert werden soll. Seit Android Version 4.0 gibt es außerdem Switches, die eine konkrete On/Off-Funktionalität symbolisieren.

4.1.7 Spinner

Ein Spinner ist ein abstrakteres GUI-Element. Es repräsentiert im Prinzip die Funktionalität eines Drop-Down-Menüs. Der Inhalt wird mit Hilfe eines Adapters definiert.

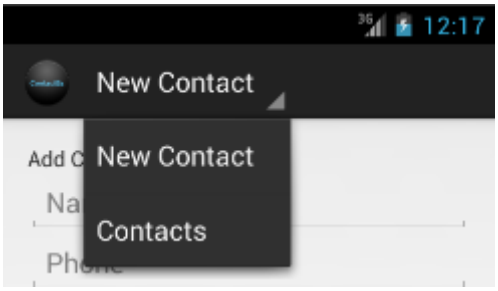

Abbildung 10 – Spinner als Menü

In Abbildung 10 ist ein Spinner zu sehen der in dem Beispiel als ein Auswahl-Menü in einer ActionBar fungiert.

4.1.8 Picker

Picker werden verwendet um die Eingabe des Benutzers auf fest definierte Werte einzuschränken z.B. Zeit- oder Datumswerte. Das Android System bietet bereits vordefinierte Picker-Elemente, die als Solche vom Entwickler ohne größeren Aufwand genutzt werden können. Das hat den Vorteil, dass Picker im Android-Umfeld einheitlicher aussehen.

- Der TimePicker ist ein GUI-Element das dazu dient, Zeitangaben vom Benutzer abzufragen.
- Das DatePicker-Element ist für Datumseingaben gedacht

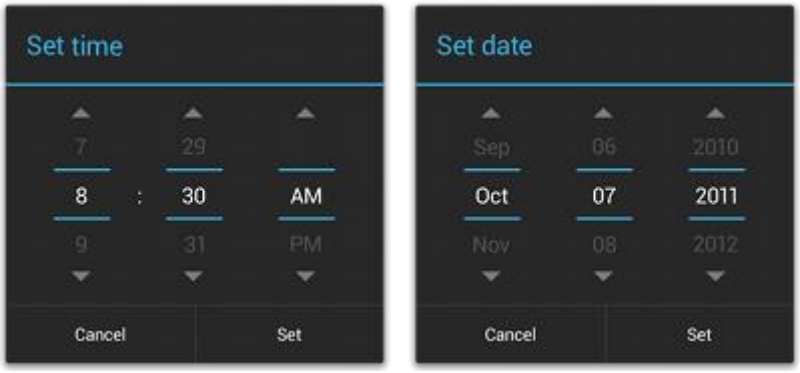

Abbildung 11 – TimePicker und DatePicker[7]

Picker-Elemente sollten, da sie viel Platz in einer Ansicht einnehmen können und nur für die konkrete Auswahl benötigt werden, in einem eigenen Dialog angezeigt werden, und zwar dann wenn sie benötigt werden.

[7] Vgl. Pickers – Android Developers Guides

4.2 Sonstige Elemente

4.2.1 Dialoge

Wie bereits erwähnt sollten Elemente die nur eine kurze Anzeigedauer haben, wie beispielsweise ein Picker, nur angezeigt werden wenn sie benötigt werden. Für solche Aufgaben sind Dialoge verantwortlich. Dialoge sind Fragmente die über eine Activity gezeichnet werden. Während ein Dialog angezeigt wird, ist die Activity im Hintergrund abgedunkelt und weiterhin sichtbar. Um Dialoge anzuzeigen muss die Klasse DialogFragment erweitert werden.

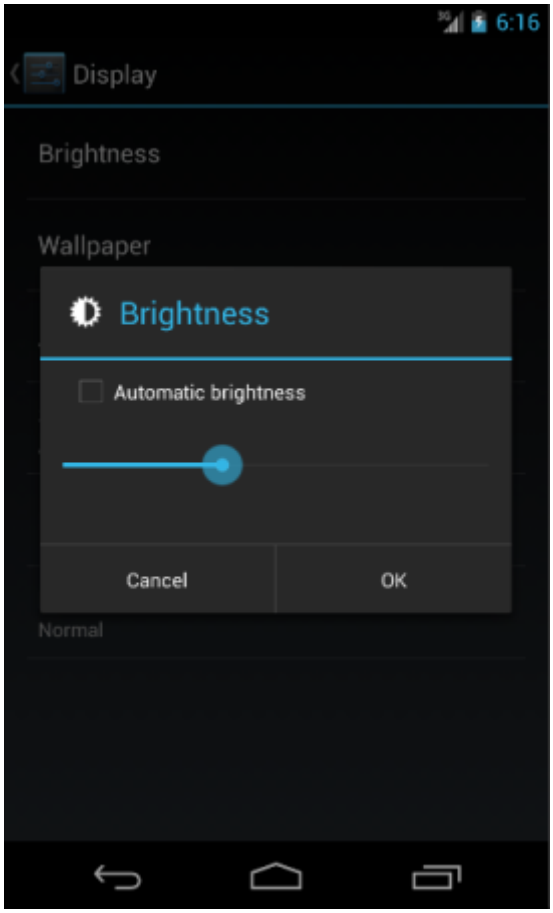

Abbildung 12 – Dialog für Helligkeitseinstellungen

4.2.2 Menus

Ein Menü ist eine häufig genutzte GUI-Komponente, die in vielen Applikationen zu finden ist. Das Verhalten sollte dabei bei allen Geräten ähnlich sein. Seit Android-Version 3 (API-Level 11) werden Menüs nichtmehr über den „Menü“-Hardware-Button angesteuert, sie sind nun Bestandteil einer GUI innerhalb der ActionBar.

Ein Menü ist ein Dialog, meist in Form einer Liste, mit dem der Benutzer eine Optionen-Activity starten kann oder durch die Applikation navigieren kann. Ein Menü wird als Ressource in einer XML-Datei definiert.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/action_settings"
        android:showAsAction="collapseActionView"
        android:title="@string/action_settings"/>
    <item
        android:id="@+id/change_activity"
        android:showAsAction="collapseActionView"
        android:title="@string/show_contacts"/>
</menu>
```

Listing 9 – Menu Beispiel

Items in einem Menü lassen sich auch in Gruppen zusammenfügen, wenn sie als Kind-Elemente eines <group>-Tags einem Menü eingefügt werden. Damit lassen sich Sichtbarkeiten oder die Aktivschaltung der gruppierten Elemente leichter steuern.

4.3 Benachrichtigungen

Benachrichtigungen werden als Notifications bezeichnet. Notifications sind nur indirekt ein Element einer GUI, sie dienen dazu dem Benutzer außerhalb einer Activity eine Benachrichtigung zu zeigen. Sie werden vom systemweiten NotificationManager verwaltet. Um eine Notification zu verwenden muss zunächst ein Builder initialisiert werden. Mit dem Builder lassen sich Notifications bauen und über den NotificationManager mit der Methode „notify“ wird dann die Notification in der NotificationBar des Geräts angezeigt. Den NotificationManager bekommen wir mit Aufrufen der Methode `getSystemService(Context.NOTIFICATION_SERVICE)`.

```
noteBuilder = new NotificationCompat.Builder(this);
noteBuilder.setContentTitle(getString(R.string.notification_title));
noteBuilder.setContentText(getString(R.string.notification_text));
noteBuilder.setSmallIcon(R.drawable.ic_launcher);
```

Listing 10 – Notification-Builder

In Listing 10 wird ein Notification-Builder initialisiert und die entsprechenden Attribute für unsere Notification werden definiert. Wird nun die Methode notify des NotificationManager-Objektes aufgerufen, übergeben wir das Notification-Objekt das wir erhalten sobald wir die build-Methode des Builders aufrufen.

Eine Notification lässt sich seit Android Version 4.2 erweitern, diese Notifications werden „Big Notification“ genannt.

Eine weitere Form von Benachrichtigungen sind Toasts. Dabei handelt es sich um eine kleine Popup-Nachricht die als kurzes Feedback für den Benutzer dient, z.B. die Information, dass ein Kontakt im Adressbuch erfolgreich gespeichert wurde.

5 Input Events

Allgemein können Events auf zwei Arten gestartet werden, zum einen können Events direkt in der Komponente in einer Layout-Datei deklariert werden und Komponenten können im Quelltext mit Listenern zu versehen werden. In der Layout-Datei kann ein Event als Attribut einer Komponente gesetzt werden, jedoch ist dabei nur ein onClick-Event möglich.

```
<Button
      android:id="@+id/b_cancel"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/cancel"
      android:onClick="buttonCancelClicked" />
```

Listing 11 – onClick-Event

Der Wert des Attributs android:onClick ist der Name der Methode die beim Klick auf die Komponente ausgeführt wird. Im Beispiel von Listing 11 ist das die Methode buttonCancelClicked. Die Methode, die in dem Attribut spezifiziert wurde, muss in der Activity vorhanden sein und einen Übergabe-Parameter vom Typ View beinhalten, dieser Parameter ist beim Aufruf der Methode die geklickte Komponente. Input Events werden vom System so gehandhabt, dass bei einer Interaktion die gewählte GUI-Komponente an die Handler-Methode übergeben wird. Diese Vorgehensweise Events einzufangen ist allerdings nur für kleinere Events geeignet, da hier nur eine Methode fix definiert werden kann und keine anderen Events, abgesehen vom onClick-Event, möglich sind.

5.1 EventListeners

Eine andere und üblichere Methode Events einzufangen ist das Verwenden von Listenern. Ein Listener wird programmatisch im Quellcode der Activity definiert, durch eine Set-Methode der Komponente kann sich diese an einem Listener registrieren. Das Beispiel von vorher würde in diesem Fall wie folgt aussehen:

```
bCancel.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
            MainActivity.buttonCancelClicked(v);
      }
});
```

Listing 12 – OnClickListener

Ein Listener implementiert ein ihm entsprechendes Listener-Interface. Im Beispiel, zu sehen in Listing 12, wurde ein OnClickListener als innere Klasse implementiert, dessen onClick-Methode die Methode aus dem Beispiel vorher aufruft.

5.2 Events

In der folgenden Liste sehen wir eine Auswahl an möglichen Events, die für das Event-Handling gängig und relevant sind.

- OnClick – beim einfachen Klick auf eine Komponente
- LongClick – beim langen Klick bzw. geklickt Halten einer Komponente
- OnTouch – anders als beim OnClick wird der Event gestartet sobald die Komponente berührt wurde.
- FocusChange – wenn sich der Focus der Komponente ändert.
- OnChange – beim Ändern des Inhalts einer Komponente, z.B. könnte ein Button erst aktiv sein, wenn zuvor ein Textfeld ausgefüllt wurde.

6 Kommunikation

Um zwischen Applikationen zu kommunizieren werden Intents verwendet. Ein Intent beinhaltet Informationen über die Intention, daher auch der Name Intent, einer Nachricht. Ein Intent kann z.B. eine Activity oder einen Dialog starten. Bei Intents wird zwischen expliziten und impliziten Intents unterschieden.

6.1 Explizite Intents

Ein expliziter Intent dient dazu eine konkrete Komponente aufzurufen. Es wird als Parameter die Zielkomponente angegeben. Da die Zielkomponenten anderer Applikationen dem Entwickler in der Regel unbekannt sind, werden die expliziten Intents für die Kommunikation innerhalb einer Applikation verwendet, beispielsweise um eine andere Activity oder einen Service zu starten. Um eine Activity zu starten, muss uns die Activity-Klasse bekannt sein. Sollte die Activity die wir starten möchten nicht in der Manifest-Datei bekannt sein, kann diese auch nicht gestartet werden.

```
Intent nachricht = new Intent(v.getcontext, SecondActivity.class);
startActivity(nachricht);
```

Listing 13 – Expliziter Intent

6.2 Implizite Intents

Bei dem Beispiel in Listing 14 handelt es sich um einen impliziten Intent, der anhand der mitgelieferten Intent-Informationen vom System einer Applikation zugeordnet werden kann und diese Applikation die angeforderte Aktion ausführen kann.

```
Intent i = new Intent(Intent.ACTION_DEFAULT,
      ContactsContract.Contacts.CONTENT_URI);
startActivity(i);
```

Listing 14 – Impliziter Intent

Um einen impliziten Intent einfangen zu können, muss in unserer Manifest-Datei ein Intent-Filter für unsere Applikation definiert sein. Ein impliziter Intent wird auch beim Starten einer App über den Launcher abgesetzt. Deshalb muss für unsere MainActivity-Klasse ein <intent-filter>-Tag als Kind-Element in der Manifest-Datei vorhanden sein, damit der Launcher unsere Activity kennt und starten kann.

7 Erste GUI entwickeln

In diesem Kapitel möchten wir abschließend das bisher Gelernte einmal in die Praxis umsetzen. Wir werden uns nun einige der erläuterten Techniken an einem konkreten Beispiel genauer ansehen.

7.1 Eclipse Projekt anlegen

Um Schrittweise vorzugehen legen wir zunächst mit Eclipse ein Android Application Projekt an. Das Android Eclipse-Plugin bietet einen Wizard zum Erstellen eines Projektes. Diesen wollen wir nutzen um die Grundstruktur unseres Projektes zu erhalten. Wir können uns an dieser Stelle eine Activity erstellen lassen, doch dies möchten wir im folgenden Beispiel selbst übernehmen. Zur Grundstruktur gehören unter anderem der Ressourcen-Ordner „res“ und die AndroidManifest.xml-Datei.

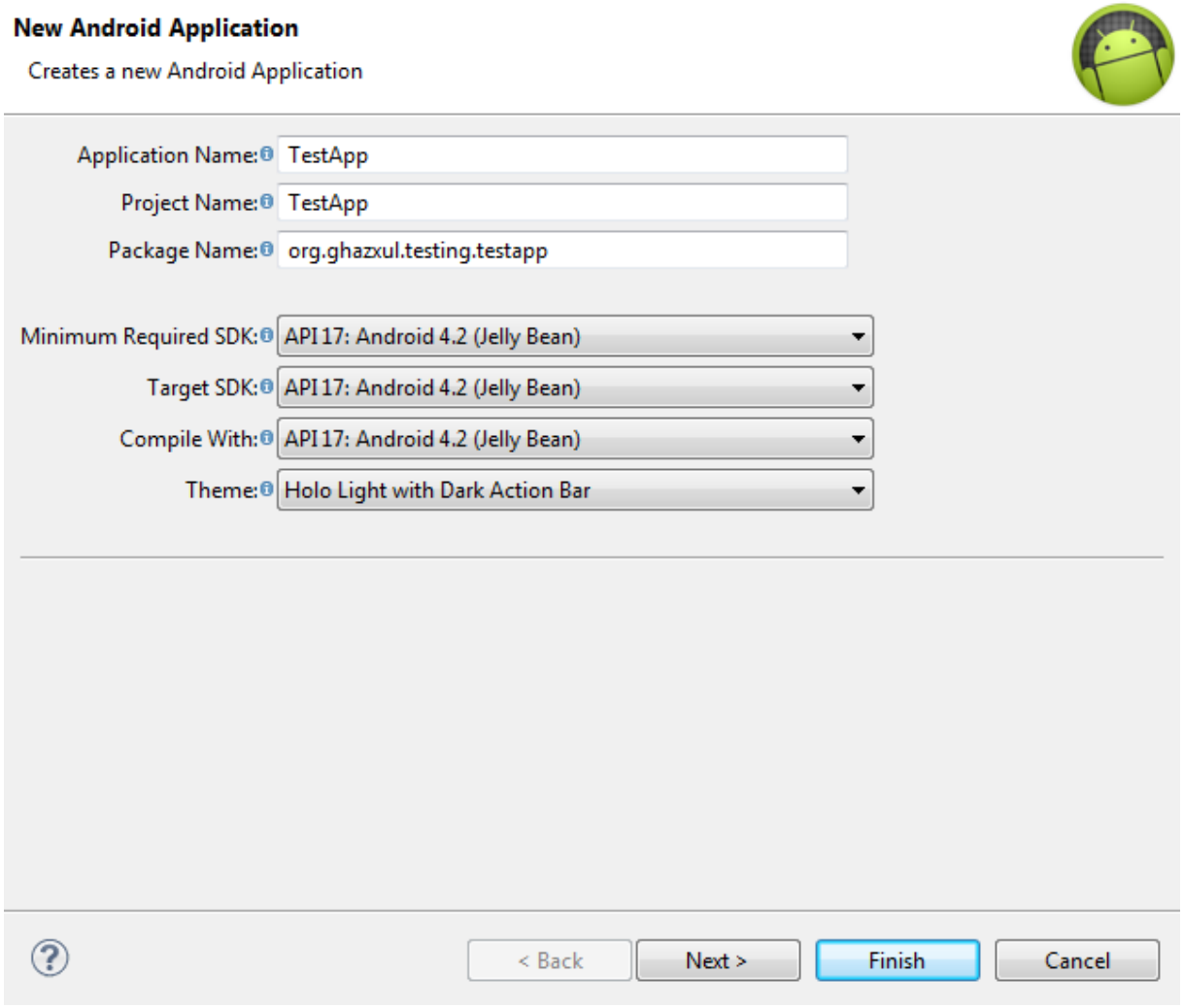

Abbildung 13 – Android Application - Project Wizard

In Abbildung 13 sehen wir den angesprochenen Wizard, hier können wir den App-Namen Projektnamen, Package-Pfad und das SDK unserer App festlegen. Nach einem Klick auf Next

können wir uns neben einer Activity und einem Launcher-Icon auch den Projektpfad im Dateisystem erstellen lassen. Wir möchten uns keine Activity erzeugen lassen, da wir diese manuell anlegen möchten. Um ein Launcher-Icon zu besitzen können wir uns hier eines selbständig generieren lassen.

Nach dem wir das Projekt angelegt haben müsste es nun im Package Explorer aussehen wie in Abbildung 14 zu sehen ist.

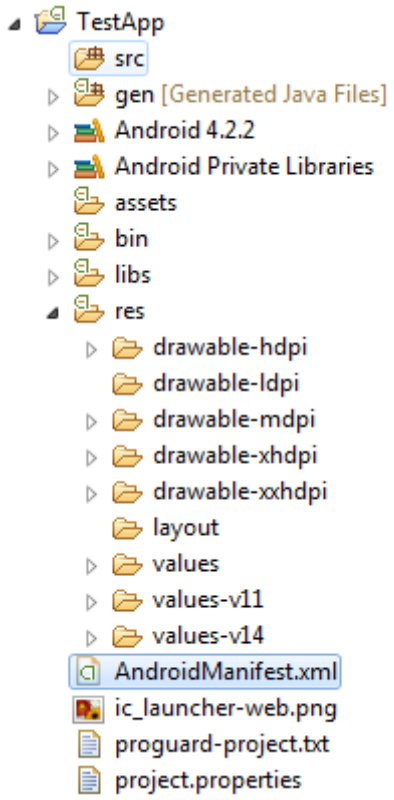

Abbildung 14 – Package Explorer

Sehen wir uns nun einmal den Inhalt der Manifest-Datei an. Diese sollte bislang kaum Informationen enthalten. Zu beachten gilt der <application>-Tag, hier wird als nächstes unsere Activity als Kind-Element eingefügt.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.ghazxul.testing.testapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="17"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
    </application>

</manifest>
```

Listing 15 – Angelegte Manifest-Datei

Wir sehen in der Manifest-Datei außerdem den zuvor definierten API-Level 17.

7.2 Activity inklusive Layout erstellen

Nun legen wir unsere eigene Activity an. Wir benötigen dazu eine Klasse in unserem Source-Ordner die die abstrakte Klasse Activity und die zugehörige Methode onCreate implementiert. Das Ergebnis sollte vergleichbar mit der in Listing 16 zu sehenden Klasse sein.

```
public class MainActivity extends Activity{
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

	}
}
```

Listing 16 – Activity-Klasse wurde angelegt

Damit die Activity vom System erkannt wird, muss nun die Activity in der Manifest-Datei definiert werden. Dazu fügen wir ein <activity>-Tag als Kind-Element des <application>-Tags ein und damit unsere App vom Launcher erkannt werden kann, müssen wir einen Intent-Filter für den Launcher hinzufügen.

```
<activity
	android:name="org.ghazxul.testing.MainActivity"
	android:label="@string/app_name">
	<intent-filter>
		<action android:name="android.intent.action.MAIN" />
		<category android:name="android.intent.category.LAUNCHER" />
	</intent-filter>
</activity>
```

Listing 17 – Activity in AndroidManifest.xml festgelegt

Nachdem wir nun eine Activity erstellt haben und diese im Manifest registriert wurde, müssen wir nun ein Layout als Content unserer Activity festlegen. Wir erstellen eine XML-Datei, mit dem Namen activity_main.xml, im Ordner res/layout/ und definieren in dieser Datei ein LinearLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical" >

</LinearLayout>
```

Listing 18 – LinearLayout für unsere MainActivity Klasse

Damit unsere Activity das Layout anzeigt müssen wir es der Activity noch zuweisen, das wird in der onCreate-Methode unserer Activity mit dem Methodenaufruf `setContentView(R.layout.activity_main);` getätigt. Ab diesem Zeitpunkt kann unsere App bereits auf einem Gerät ausgeführt werden.

7.3 Komponenten einfügen

Nun wollen wir einige der kennengelernten Komponenten in unser Layout einfügen. Wir haben uns überlegt eine Applikation zu schreiben, die die primitive Funktionalität bietet einen Text zu erfassen und diesen auf verschiedene Arten zu verarbeiten. Aus diesem Grund benötigen wir ein TextView zwei Textfelder, eine RadioGroup und einige Buttons die wir nun in unser Layout einfügen.

Die Auswahl, die in der Radio-Group getroffen wurde, soll darüber entscheiden, welche Aktion ausgeführt oder gestartet werden soll. Dem Benutzer soll die Möglichkeit geboten werden den angegebenen Text in einem Textview anzuzeigen, eine Notification anzuzeigen oder eine Activity zu starten.

Für die Beschriftungen der Elemente legen wir einige Strings in der strings.xml-Datei fest.

```
<TextView
        android:id="@+id/textview"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="16dp"
        android:paddingTop="16dp"
        android:text="@string/hello_world" />
<EditText
        android:id="@+id/textfeld1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/hint_tf1" />
<EditText
        android:id="@+id/textfeld2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/hint_tf2" />

<RadioGroup
        android:id="@+id/radio_group"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" >
        <RadioButton
            android:id="@+id/radio_notification"
            android:text="@string/radio_notification" />
        <RadioButton
            android:id="@+id/radio_textview"
            android:text="@string/radio_textview" />
        <RadioButton
            android:id="@+id/radio_activity"
            android:text="@string/radio_activity" />
</RadioGroup>

<RelativeLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right" >
    <Button
            android:id="@+id/button_ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_alignParentTop="true"
            android:text="@string/button_ok" />
    <Button
            android:id="@+id/button_cancel"
```

Listing 19 – Komponenten im LinearLayout

Wir haben zusätzlich zu den einzelnen Komponenten ein RelativeLayout hinzugefügt, welches die genannten Buttons beinhaltet und haben sie nebeneinander angeordnet. Unsere App sollte nun aussehen, wie in Abbildung 15 zu sehen ist.

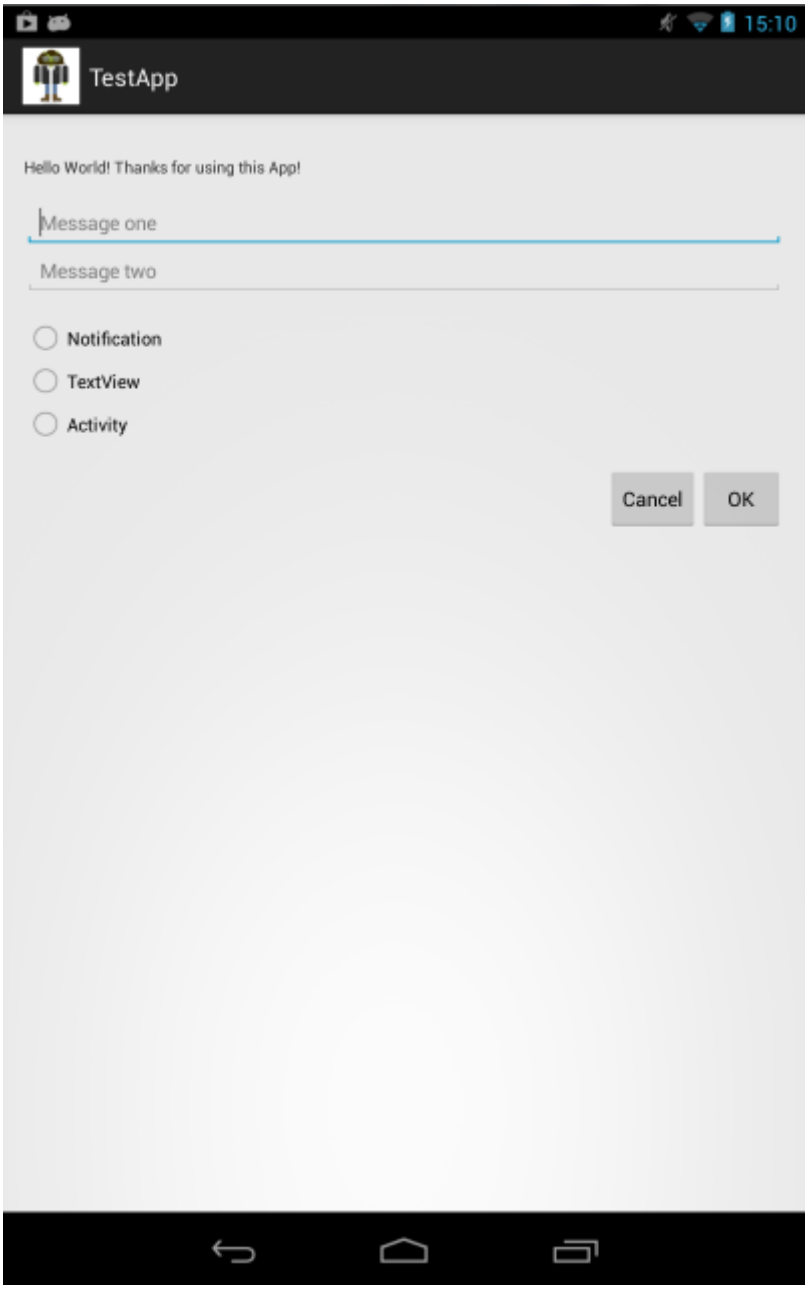

Abbildung 15 – Unsere Test-App

7.4 Handling

Um das Handling der Radiobuttons zu realisieren, müssen wir für unsere Activity eine Überlegung machen, wie dies geschehen soll. Dafür müssen Vorbereitungen getroffen werden. Wir müssen einen Parameter bestimmen der uns innerhalb unserer Activity sagen kann welcher Radiobutton ausgewählt wurde, dafür verwenden wir am besten eine Integer-Variable, der die Komponenten-ID des ausgewählten Radiobuttons zugewiesen wird. Das Vorgehen hat den Vorteil, dass der Radiobutton über die ID eindeutig identifizierbar ist und deshalb auch unsere Variable immer eindeutig sagen kann welcher Radiobutton ausgewählt wurde.

7.4.1 onClick-Attribut

Zudem benötigt die Activity-Klasse eine Methode die ausgeführt werden soll, sobald ein Radiobutton gewählt wurde. Am einfachsten funktioniert das mit dem onClick-Attribut einer Komponente. Wir fügen als nächstes bei den Radiobuttons das Attribut android:onClick mit dem Wert „onRadioButtonClicked“ ein, welcher unser Methodenname sein soll. In unserer Activity-Klasse implementieren wir nun diese Methode mit dem Rückgabewert void und einem Übergabeparameter vom Typ View. Diese Methode soll nun für das Radiobutton-Handling verantwortlich sein wird und sollte wie folgt aussehen.

```
public void onRadioButtonClicked(View view) {

	boolean clicked = ((RadioButton) view).isChecked();

	switch (view.getId()) {
	case R.id.radio_notification:
		if (clicked)
			checkedValue = R.id.radio_notification;
		break;
	case R.id.radio_textview:
		if (clicked)
			checkedValue = R.id.radio_textview;
		break;
	case R.id.radio_activity:
		if (clicked)
			checkedValue = R.id.radio_activity;
		break;
	default:
		break;
	}
}
```

Listing 20 – onClick-Handling für die Radiobuttons

checkedValue ist die genannte Integer-Variable, welche in der Activity-Klasse als private field bekannt ist.

7.4.2 Listener

Das Handling unserer Buttons sollen Listener übernehmen. Dafür gehen wir in den Quelltext der Activity-Klasse und initialisieren die Button-Komponenten in der von der abstrakten Activity-Klasse geerbten onCreate-Methode.

```
buttonOk = (Button) findViewById(R.id.button_ok);
buttonOk.setOnClickListener(new ButtonOKListener());

buttonCancel = (Button) findViewById(R.id.button_cancel);

buttonCancel.setOnClickListener(new ButtonCancelListener());
```

Listing 21 – Buttons initialisiert

Die beiden Listener-Klassen implementieren dabei das Interface OnClickListener und sind in unserem Konstrukt nur innere Klassen mit private-Zugriffsschutz. Nachfolgend sind diese beiden Listener Klassen zu sehen.

```
private class ButtonCancelListener implements OnClickListener {
	@Override
	public void onClick(View v) {
		finish();
	}
}
```

Listing 22 – ButtonCancelListener-Klasse

Einziger Zweck dieses Listeners ist es die Activity zu beenden, sobald der Benutzer auf den Cancel-Button klickt. Ein Activity-Lebenszyklus wird mit dem Aufrufen der finish-Methode beendet.

Ein wenig interessanter wird der Listener für unseren OK-Button, dieser soll dazu dienen die von uns gewünschte Funktionalität umzusetzen. Dies realisieren wir nun über die zuvor bestimmte *checkedValue* Variable. Der Listener soll in diesem Fall nur die Aufgaben delegieren und die entsprechende Funktionalität aufrufen, die wir benötigen.

```
private class ButtonOkListener implements OnClickListener {

	@Override
	public void onClick(View v) {

		switch (checkedValue) {
		case R.id.radio_notification:
			startNotification();
			break;
		case R.id.radio_textview:
			startTextviewChange();
			break;
		case R.id.radio_activity:
			startSecondActivity(v.getContext());
			break;
		default:
			break;
		}
	}
}
```

Listing 23 – ButtonOkListener-Klasse

Wir starten mit jeder möglichen Radiobutton-Option nun eine andere Funktionalität.

7.4.3 Funktionalität hinterlegen

Damit wir nun auch tatsächlich ein Ergebnis bekommen muss unseren Methoden auch Funktionalität hinterlegt werden. Dabei ist zu beachten, dass in den folgenden Code-Beispielen keine Behandlung von möglichen Falscheingaben zu sehen ist, um die Beispiele einfach zu halten und nur das wesentliche zu zeigen. Obwohl diese immer berücksichtigt werden sollten, wenn Benutzerschnittstellen implementiert werden.

Die Methode startTextviewChange soll die Textview in unserem Layout bearbeiten. Es soll der Text eingefügt werden, der in die Textfelder eingegeben wurde. Dazu müssen die Textfelder deklariert und initialisiert worden sein. Der Text einer Textview einfach mit der Methode setText geändert werden. Der Inhalt unserer Methode könnte so aussehen:

```
public void startTextviewChange() {
	textView.setText(textfield1.getText().toString() + " "
			+ textfield2.getText().toString());
}
```

Listing 24 – startTextviewChange-Methode

Diese Funktionalität ist mit Abstand die einfachste. Als Nächstes werden wir eine Notification mit unserem eingegebenen Text anzeigen lassen. In der startNotification-Methode soll diese Funktionalität nun hinterlegt werden. Doch zuerst müssen wir in der onCreate-Methode einen Notification-Builder und den Android-NotificationManager initialisieren.

```
builder = new NotificationCompat.Builder(this);
manager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
```

Listing 25 – Initialisierung von Notification-Builder und NotificationManager

Nachdem nun Builder und Manager initialisiert sind, können wir in unserer startNotification-Methode dem Builder den Inhalt unserer Textfelder zuweisen und eine Notification anzeigen lassen.

```
public void startNotification() {
	builder.setContentTitle(textfield1.getText().toString());
	builder.setContentText(textfield2.getText().toString());
	builder.setSmallIcon(R.drawable.ic_launcher);

	manager.notify(0, builder.build());
}
```

Listing 26 – startNotification-Methode

Nun starten wir unsere App und probieren die implementierten Funktionalitäten aus. Wir sollten nun eine Notification angezeigt bekommen, mit dem eingegeben Text als Titel bzw. Nachrichtentext (Abbildung 16).

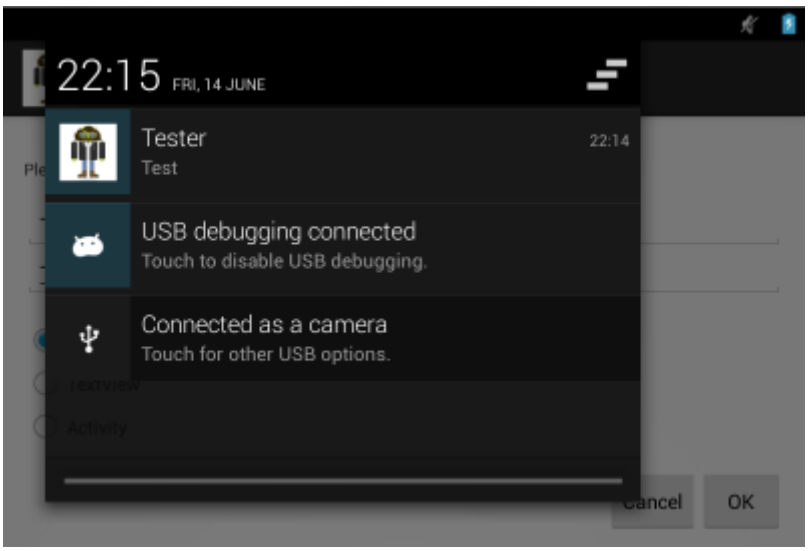

Abbildung 16 – Notification unserer TestApp

Als nächstes möchten wir uns ansehen wie eine andere Activity mit einem expliziten Intent gestartet wird. Doch bevor wir unsere startSecondActivity-Methode implementieren, müssen wir erst eine weitere Activity anlegen. Wir können dazu einen Wizard verwenden oder wie zuvor in Kapitel 7.2 beschrieben vorgehen. Für diese Demonstration benötigen wir kein kompliziertes Activity-Layout, es reicht ein einfaches LinearLayout mit einer Textview. Wir sollten dabei nicht vergessen, dass wir dieser Activity in der onCreate-Methode ein Layout zuzuweisen und, besonders wichtig, dass wir die Klasse in der Manifest-Datei bekannt machen. Nun können wir auch schon in der startSecondActivity-Methode die Activity starten.

```
public void startSecondActivity(Context context) {
      startActivity(new Intent(context, SecondActivity.class));
}
```

Listing 27 – startSecondActivity-Methode

Starten wir nun die App und wählen den Radiobutton „Activity“ startet sich unsere zweite Activity, nachdem wir auf OK geklickt haben.

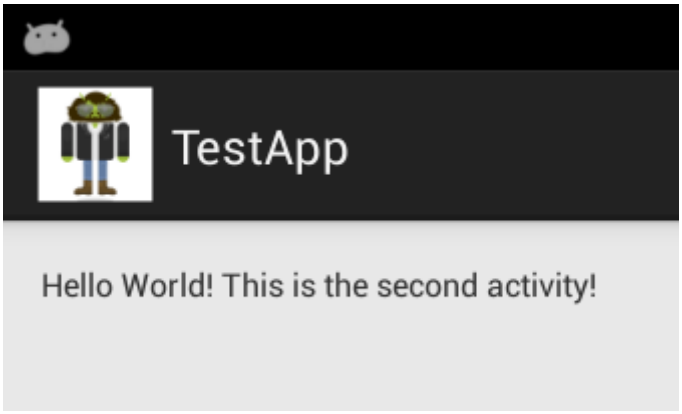

Abbildung 17 – Zweite Activity unserer TestApp

7.5 Alternative Ressourcen hinzufügen

Wir möchten unserer App nun einige alternative Ressourcen zuweisen und verwalten. In diesem Kapitel werden wir ein Alternatives Layout anlegen, damit unsere App auch im Querformat gut aussieht. Außerdem werden wir eine Lokalisation für den deutschsprachigen Raum unserer App hinzufügen.

7.5.1 Layout für Querformat

Um ein Layout für den Querformat zu definieren muss eine Layout Datei im res/layout-land/ (landscape-Layout) angelegt werden, die einen identischen Namen wie das MainActivity-Layout hat, also activity_main.xml. Nun können wir den Inhalt unseres vorherigen Layouts beliebig umstellen, bis es unseren Ansprüchen genügt und dabei bewusst die Layout-Hierarchie ein wenig aufblasen.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:padding="16dp" >

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/textview"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:paddingBottom="16dp"
            android:paddingTop="16dp"
            android:text="@string/hello_world" />

        <EditText
            android:id="@+id/textfeld1"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:hint="@string/hint_tf1" />

        <EditText
            android:id="@+id/textfeld2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:hint="@string/hint_tf2" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <RadioGroup
            ... >
                ...
        </RadioGroup>

        <RelativeLayout >
                ...
        </RelativeLayout>
    </LinearLayout>
</LinearLayout>
```

Listing 28 – Landscape-Layout

Starten wir nun unsere App erneut, werden wir bemerken, dass sich diese aufgepumpte Layout-Hierarchie auf die Performance auswirkt. Die Benutzeroberfläche wird nun unter Umständen bedeutend länger brauchen, bis sie gezeichnet ist. Das hängt natürlich auch von der Hardware unseres Gerätes ab. Aus diesem Grund wurden für die Android-GUIs die Fragmente eingeführt, die den Vorteil haben, dass sie beim Layout-Wechsel nicht komplett gezeichnet werden müssen, sondern nur neu angeordnet werden können.

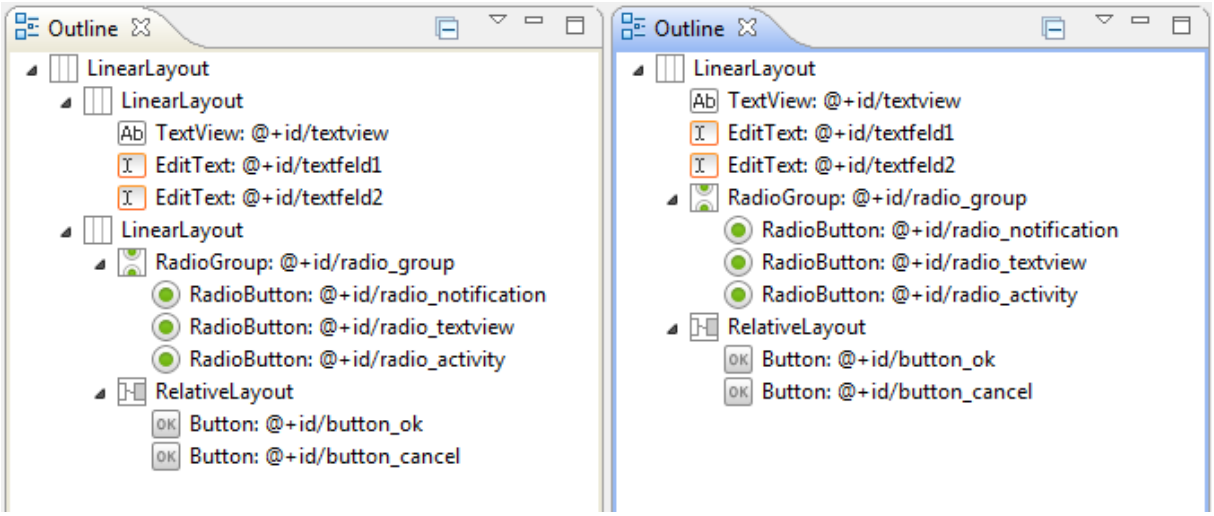

Abbildung 18 – Layout Hierarchie in der Outline-View

In Abbildung 18 sehen wir die beiden Layout-Hierarchie-Bäume in der Outline-View von Eclipse (links: landscape, rechts: default).

7.5.2 Lokalisation für deutschsprachige Geräte

Um unsere App auch für Benutzer zu optimieren, die eine andere Sprachausgabe auf ihrem Gerät eingestellt haben, werden verschiedene Lokalisationen definiert. Wir möchten nun die Sprachausgabe unserer App auch für deutschsprachige Geräte zur Verfügung stellen. Um das zu realisieren legen wir einen Ressourcen-Ordner mit dem Namen res/values-de/ an und kopieren die strings.xml-Datei aus dem res/values/ Ordner in den neu angelegten Ordner. Nun können wir die res/values-de/strings.xml öffnen und die darin enthaltenen Texte übersetzen. Abschließend stellen wir die Sprachausgabe des Geräts auf Deutsch – falls dies nicht bereits geschehen ist – und starten unsere App.

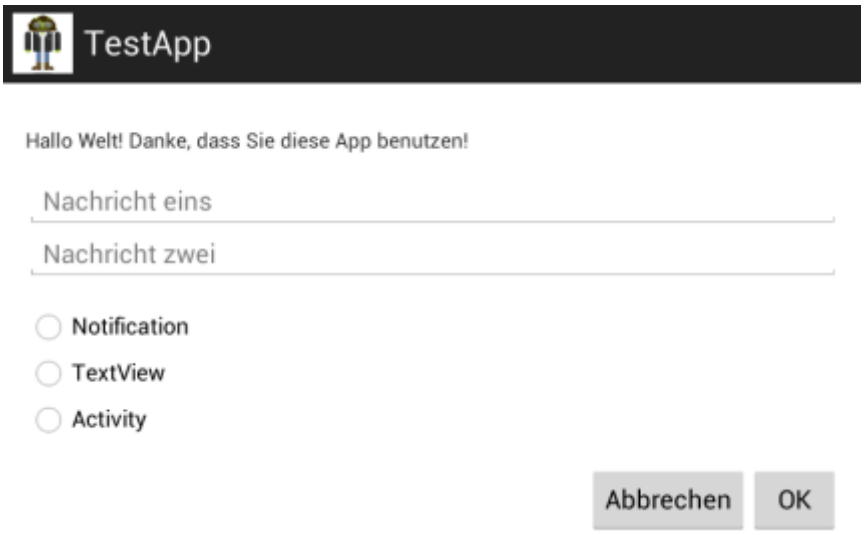

Abbildung 19 – Deutsche Lokalisation der Beispiel-App

So lässt sich einfach und flexibel die geschriebene App übersetzen. Ein Problem beim Übersetzen könnte sein, dass die Übersetzung für den jeweiligen Kontext geeignet sein muss. In unserem Beispiel ist das noch recht überschaubar, aber bei größeren und komplexeren Applikationen kann eine Übersetzung viel Zeit kosten.

8 Ausblick und Fazit

Dieses Dokument hat uns einen Überblick über die vielen Möglichkeiten beim Entwickeln einer Android-GUI verschafft und eine Handvoll der vielen Komponenten einer Android-Applikation wurden anhand eines – wenn auch sehr primitiven – Beispiels näher gebracht. Für künftige Applikationen gilt zu beachten, dass dieses Dokument „nur“ die Grundlagen der GUI-Programmierung erläutern soll und der Spielraum für Android-Apps bedeutend größer ist, da es unzählige Komponenten und Elemente gibt, die hier nicht besprochen wurden. Beispielsweise kann die Verwendung von Broadcast Receivern oder Services das Spektrum an Vielfalt einer Applikation bedeutend erweitern.

Da hinter dem Android-Betriebssystem eine sehr große und mächtige Community steht und die API für Android-Applikationen ausgesprochen gut dokumentiert ist, hat man als Entwickler sehr gute Chancen sehr schnell zu brauchbaren Ergebnissen zu kommen.

Ein großer Nachteil ist, dass für ältere Geräte und damit auch alte Android-Versionen eine gewisse Abwärtskompatibilität gegeben sein muss, um möglichst viele Benutzer ansprechen zu können. Da viele End-Nutzer veraltete Geräte, mit Hardware aus der vorletzten Generation, verwenden, ist es nicht immer einfach auch für diese Benutzer seine Applikation zur Verfügung zu stellen. Viele Komponenten sind erst seit Android Version 3 oder später verfügbar und können nur durch das Einbinden von zusätzlichen Libraries auf Geräten mit einer älteren Version verwendet werden. Beispielsweise das Nutzen von Dialogen und Fragmenten wird erst seit Version 3.0 (API-Level 11) unterstützt.

Dem Autor dieses Dokuments hat es besonders viel Spaß gemacht, die unglaublich große Vielfalt, die in einem so kleinen Gerät stecken kann, zu entdecken und auch kleinere Applikationen einmal auf dem eigenen Gerät auszuprobieren. Zudem ist es interessant zu erfahren, wie das Gerät arbeitet und welche Mühe in jedem noch so kleinen Detail steckt.

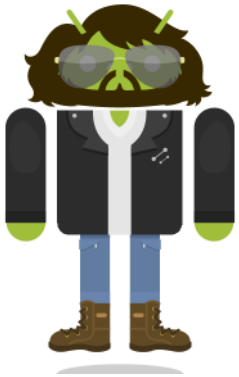

Abbildung 20 – Ein Android[8]

[8] Bild wurde erstellt mit der App „Androidify“

Literaturverzeichnis

Android Developers Guides – http://developer.android.com/	[Datum: 27.05.2013]
Thomas Künneth – Android 4 – Galileo Computing	[2. Auflage, 2012]
Christian Bleske – Java für Android – Franzis Verlag	[1. Auflage, 2012]
DroidWiki – http://www.droidwiki.de/	[Datum: 01.06.2013]