

Kursunterlagen

Modul 335 Mobile-Applikationen realisieren

Dokumentinformationen

Dokumenttitel: Kursunterlagen M335 R6
Thema: Modul 335: Mobile-Applikationen realisieren
Dateiname: ku-modul335v4.0.docx
Speicherdatum: 18.04.2018
Autor: Joel Holzer

Die vorliegenden Kursunterlagen wurden in ihrer Originalfassung von der Noser Young Professionals AG (NYP) entwickelt. Der Einsatz und die Weiterentwicklung dieser Unterlagen in der Noser Young Professionals AG erfolgt mit der ausdrücklichen Genehmigung des NYP.

Inhaltsverzeichnis

1	Einleitung	4
1.1	Sinn und Zweck	4
1.2	Referenzdokumente und Internetquellen.....	4
2	Administratives.....	5
2.1	Checklisten zur Selbstkontrolle	5
2.2	Verwendete Software.....	7
2.3	Wichtige Webseiten	8
2.4	Code-Beispiele / Demo-Projekte	9
3	App-Typen im Überblick.....	10
3.1	Native App	10
3.2	Web App	13
3.3	Hybride App	15
3.4	Native, Web und Hybrid App – Welcher App-Typ soll ich wählen?	17
3.5	Cross Plattform Entwicklung	18
4	Android.....	19
4.1	Entstehung und historische Entwicklung	20
4.2	Systemarchitektur	22
4.3	Android SDK	24
4.4	Android Virtual Device Manager.....	25
4.5	Gradle als Build Management Tool	28
5	Das erste eigene Android-Projekt	29
5.1	Projekt erstellen	29
5.2	Projektstruktur.....	31
5.3	App im Emulator ausführen.....	35
5.4	App auf Smartphone übertragen	35
5.5	Programmcode debuggen.....	36
6	Benutzeroberfläche gestalten und realisieren	37
6.1	Design – Grundlagen	37
6.2	Benutzeroberfläche für Android realisieren	41
6.3	Benutzeroberfläche optimieren	52
7	Dynamische Benutzeroberflächen	58
7.1	Activity Lifecycle.....	58
7.2	Fragment Lifecycle.....	60
7.3	Activity-XML mit Java-Klasse verbinden.....	62
7.4	Benutzereingaben verarbeiten	63
7.5	Neue Ansicht starten.....	70
7.6	Dialoge	74
7.7	Spinners (Dropdowns)	75
7.8	ListView	78
8	Daten persistieren	79
8.1	Daten in Datei persistieren	79
8.2	Daten in Datenbank persistieren	86
9	Zugriff auf Webservices	91
9.1	Was ist ein Webservice?	91
9.2	SOAP-Webservices	92

9.3	REST-Webservices.....	93
9.4	REST-Webservices testen	94
9.5	Zugriff auf REST-Webservices in Android-App.....	95
10	Sensoren.....	96
10.1	Sensoren bei Android.....	96
10.2	Programmierung mit Sensoren in der eigenen App.....	97
11	Multitasking	100
11.1	Threads.....	100
11.2	Worker Threads erstellen mit Thread und Runnable	101
11.3	AsyncTasks	102
12	Apps testen	103
12.1	White Box Testing von Apps	103
12.2	Black Box Testing von Apps.....	110
12.3	Testvorgehen in einem App-Projekt	113
13	Apps veröffentlichen.....	116
13.1	Google Play Store.....	116
13.2	Entwicklerkonto erstellen und Store-Eintrag vorbereiten	116
13.3	Release der App erstellen.....	118
13.4	App in Store hochladen.....	119
13.5	App im Store updaten	120
13.6	Statistiken in der Google Play Developer Console	121
14	Abbildungsverzeichnis	122
15	Tabellenverzeichnis	123
16	Quellen.....	124

Änderungsgeschichte

Version	Datum	Autor	Details
1.0	01.06.2016	holzer	Initialversion NYP
2.0	20.10.2016	holzer	Überarbeitete Version: - Versionen & Links aktualisiert
2.0 ZH	18.11.2016	holzer	Dokument für den ÜK in ZH überarbeitet.
3.0 ZH	10.01.2018	holzer	Versionen und Links aktualisiert, Codebauteile auf aktuelle Android- und Java-Versionen angepasst.
4.0	18.04.2018	Holzer	Weitere Android-Beispielprojekte hinzugefügt.

Tabelle 1 Änderungsgeschichte

1 Einleitung

1.1 Sinn und Zweck

Dieses Dokument unterstützt im Unterricht des ÜK-Moduls 335. Es kann einerseits vom Dozenten für den Unterricht in der Klasse beigezogen werden, andererseits dient es den Lernenden als Nachschlagewerk und Lernleitfaden.

Der klare Fokus des Dokuments liegt auf den Inhalten der Modulbeschreibung. Dieses Dokument beinhaltet nicht alle Bereiche der Entwicklung von mobilen Applikationen, da dies den Zeitrahmen des ÜK-Moduls bei weitem übersteigen würde. Alle im ÜK in der Theorie behandelten Themen sind jedoch Teil dieses Dokuments.

Die Entwicklung von mobilen Applikationen im Modul 335 erfolgt für Android-Smartphones mit Hilfe von Android Studio als Entwicklungsumgebung. Die Inhalte dieses Dokuments beziehen sich daher vorwiegend auf die Android-Programmierung. Alternative Technologien werden nur am Rand betrachtet.

Für die Durcharbeitung dieser Unterlage werden fundierte Kenntnisse in der Programmiersprache Java, gute Kenntnisse und korrekte Anwendung der Prinzipien der Objektorientierten Programmierung, Kenntnisse über die gängigen Design Patterns, sowie Kenntnisse in der Dokumentation von Software (UML, etc.) vorausgesetzt. Weiter wird angenommen, dass es sich bei den Lernenden um „Digital Natives“ handelt, die Benutzung des Smartphones und deren Apps damit zu deren Alltag gehört.

Die Abfolge der Kapitel in diesem Dokument entspricht dem Ablauf der Unterrichtseinheiten. Der Leser wird Kapitel für Kapitel in die Android-Programmierung eingeführt und sollte nach der Durcharbeitung dieses Dokuments, mit Unterstützung der Unterrichtseinheiten und seinen eigenen Notizen, in der Lage sein, selbstständig eine Android-App entwickeln zu können.

1.2 Referenzdokumente und Internetquellen

- [1] Modulidentifikation zum Modul 335, Modulbaukasten Release 3 (BIVO R6)
<https://cf.ict-berufsbildung.ch/modules.php?name=Mbk&a=20101&cmodnr=335>

2 Administratives

2.1 Checklisten zur Selbstkontrolle

Dieses Kapitel enthält Checklisten, mit welchen die Lernenden selbstständig überprüfen können, ob sie die notwendigen Kenntnisse für die in der Modulidentifikation definierten Handlungsziele beherrschen.

2.1.1 Checkliste allgemeine Kenntnisse zu mobilen Apps

Der Lernende kann:

- Die Unterschiede von native Apps, hybriden Apps und Web Apps aufzeigen
- Tools zur Entwicklung von native Apps für Android und iOS vorstellen
- Tools zur Entwicklung von hybriden Apps und Web Apps beim Namen nennen

2.1.2 Checkliste Design-Kenntnisse

Der Lernende kann:

- Ergonomische Standards (z.B. EN-9241-110) und deren Anwendung bei mobilen Apps mit erläutern
- Material Design von Google erläutern
- GUIs (Activity / Fragments) für unterschiedliche Bildschirmgrößen und –Ausrichtungen mit Android Studio realisieren und testen.....

2.1.3 Checkliste Android-Framework

Der Lernende kann:

- Über die Verbreitung von Android im Smartphone-Markt Auskunft geben
- Die Systemarchitektur von Android mit Android Runtime, Dalvik, ART und den Application Framework Layers in den Grundzügen erklären.....
- Den Unterschied und die Aufgabe des Virtual Device Manager und des SDK Managers aufzeigen und beide Tools bei der Realisierung von Android-Apps anwenden

2.1.4 Checkliste Android-Entwicklung

Der Lernende kann:

- Android-Studio zur Entwicklung, zum Debugging und Deployment einer Android-App anwenden
- Eine Android-Apps mit Fragments und Activities erstellen und die Unterschiede von Fragments und Activities erläutern können
- Mehrsprachige Apps entwickeln.....
- Apps mit mehreren Ansichten und GUI-Events entwickeln.....
- Logfiles in die App einbauen / Aufs Filesystem schreiben
- Daten in eine SQLite-DB persistieren unter Anwendung eines OR-Mappers
- Die wichtigsten Sensoren von Android-Smartphones erläutern.....

2.1.5 Checkliste Testing

Der Lernende kann:

- Whitebox-Tests (Unit-Tests) für Android erstellen und durchführen
- Blackbox-Tests für Android-Apps unter Einbezug der GUI-Kenntnisse erstellen und durchführen.....

2.1.6 Checkliste Veröffentlichung von Apps

Der Lernende kann:

- Kennt den Google Play Store.....
- Die nötigen Schritte zur Veröffentlichung einer App im Google Play Store aufzeigen
- Die verschiedenen Lizenzierungsmodelle im Google Play Store erläutern

2.2 Verwendete Software

Um Android-Apps einer gewissen Grösse und mit einer gewissen Funktionalität zu entwickeln, braucht es mehr als nur die Entwicklungsumgebung "Android Studio" und ein Android-Smartphones.

Nachfolgendes Kapitel erläutert kurz die wichtigsten Tools, welche bei der Entwicklung von Android-Apps hilfreich sein können. Die Programm-Beispiele in diesem Dokument basieren alle auf der Anwendung dieser Tools.

2.2.1 Android Studio

Android Studio ist die offizielle Entwicklungsumgebung (IDE) zur Entwicklung von Android-Apps. Sie wird kostenlos von Google zum Download angeboten und ständig weiterentwickelt.

Android Studio basiert auf IntelliJ. IntelliJ ist eine Entwicklungsumgebung (IDE) zur Entwicklung von Java-Applikationen und ein grosser Konkurrent von Eclipse. Die IntelliJ IDEA, wie die Entwicklungsumgebung genau heisst, ist jedoch im Gegensatz zu Eclipse kostenpflichtig. Daher verwenden viele Berufsschulen, Fachhochschulen und auch die NYP für die Java-Entwicklung Eclipse anstelle von IntelliJ.

Bis Ende 2015 war es auch mit Eclipse und deren Android Developer Tool möglich, Android-Apps zu entwickeln. Seit Ende 2015 erhält man dafür jedoch keinen offiziellen Support mehr.

Da für Android in Java entwickelt wird, setzt Android Studio zum Ausführen von Apps das Java Development Kit (JDK) voraus.

Download Android Studio: <http://developer.android.com/sdk/index.html>

Download Java SDK (Version 8):

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

2.2.2 Versionsverwaltung Programmcode (GIT)

GIT ist das im ÜK eingesetzte Tool für die Versionsverwaltung von Programmcode. Einerseits stellt der Dozent gewisse Unterlagen & Beispielprojekte über ein GIT-Repository zur Verfügung, andererseits müssen die verschiedenen Gruppen den Source Code ihrer Projekte über ein GIT-Repository austauschen und dem Dozent abgeben.

Download GIT: <https://git-scm.com/>

Download SourceTree: <https://www.sourcetreeapp.com/>

Download GitKraken: <https://www.gitkraken.com/>

2.2.3 Entwurf des GUIs

Im Rahmen der Projektarbeit müssen die Lernenden einen GUI-Entwurf ihrer App vornehmen. Dabei setzen wir Pencil mit verschiedenen Stencils ein.

Download Pencil: <http://pencil.evolus.vn/>

Download Android Lollipop Pencils: <https://github.com/nathanielw/Android-Lollipop-Pencil-Stencils/releases/tag/v1.1.0>

Download Material Icons: <https://github.com/nathanielw/Material-Icons-for-Pencil/releases/tag/v2.0.0>

2.3 Wichtige Webseiten

Nachfolgend ein paar Webseiten, welche bei der Entwicklung von Android-Apps sehr hilfreich sein können.

2.3.1 Android Training

Offizielle Webseite von Google für die Einführung in die Entwicklung mit Android. Beinhaltet gute Erläuterungen und viele Codebeispiele zu den wichtigsten Gebieten der Android Entwicklung (z.B. GUI, Speichern von Daten, Sensoren, Multimedia, etc.)

<http://developer.android.com/training/index.html>

2.3.2 Android Material Design

Offizielle Webseite von Google für das Design von Android-Apps. Richtet sich vor allem an Designer und gibt wichtige Tipps und Downloads zu Animationen, Styles, Layout, Usability, etc.

<http://developer.android.com/design/index.html>

2.3.3 EN ISO 9241-110

Folgende Webseite bietet eine gute Übersicht über die Gestaltungsgrundsätze von Dialogen (GUIs) nach ISO 9241-110.

http://www.ergo-online.de/site.aspx?url=html/software/grundlagen_der_software_ergonomie/grundsaezze_der_dialog_gestalt.htm

2.4 Code-Beispiele / Demo-Projekte

Auf dem GitHub-Repository von Joel Holzer sind nachfolgende Code-Beispiele / Demo-Projekte abgelegt. Diese sind auch jeweils in den entsprechenden Kapiteln dieses Dokuments angegeben.

<https://github.com/joe-nyp/modul-335-examples>

Projekt	Beschreibung	Kapitel
HelloWorld	Gibt „Hello World!“ auf dem Bildschirm aus.	5
LinearLayoutExample	Aufbau eines GUIs mit dem LinearLayout. Zeigt einen Text, 2 Labels, 2 Eingabefelder und 2 Buttons an.	6.2.4
RelativeLayoutExample	Aufbau eines GUIs mit dem RelativeLayout. Zeigt dasselbe GUI wie LinearLayoutExample.	6.2.4
ConstraintLayoutExample	Aufbau eines GUIs mit dem ConstraintLayout. Zeigt dasselbe GUI wie LinearLayoutExample.	6.2.4
ToastExample	Benutzereingabe aus Textfeld entgegennehmen und bei Buttonklick in einem Toast anzeigen.	7.4
IntentDemo	Starten einer zweiten Activity durch Klick auf einen Button. Übergabe von Daten von Activity1 und Activity2 mit Intent.	7.5
StaticSpinnerDemo	Spinner mit statischen Einträgen aus Arrays.xml.	7.7.1
DynamicSpinnerDemo	Spinner mit dynamischen Einträgen aus der Datenbank. Anzeige von Lernenden mit Foto und Namen.	7.7.3
ListViewExample	ListView mit dynamischen Einträgen aus der Datenbank. Anzeige der Einträge mit Titel und Nr.	7.8
LoggingExample	Loggen von Fehlermeldungen in Textfile.	8.1.4
DatabaseExample	Speichern und Lesen in SQLite-Datenbank mit Room Persistence Library	8.2.3
SensorDemo	Zeigt alle auf dem Smartphone verfügbaren Sensoren an und ermittelt den Luftdruck und zeigt diesen in einem Toast an.	10.2
JUnitDemo	Lokale und instrumented Unit Test.	12.1.3
InstrumentedUnitTestDemo	Test eines Formulars mit Instrumented Unit Tests.	12.1.5
IntegrationTestDemo	Test von Formulareingaben und Validierung mit Integration Tests.	-
CardViewExample	Anzeige von Einträgen mit Fotos in einer CardView (RecyclerView).	-
PhotoDemo	Foto schiessen mit der Kamera-App und im Filesystem ablegen	-
SharedPreferencesDemo	Speichern und Auslesen von Werten in/aus die Shared Preferences.	-

Tabelle 2 Code-Beispiele / Demo-Projekte auf GitHub

3 App-Typen im Überblick

Applikationen in der Smartphone-Welt werden als „Apps“ bezeichnet. Apps gibt es mittlerweile für praktisch jedes Anwendungsgebiet, sei es als Navigator beim Autofahren, für das Gaming im Zug oder das Lesen von E-Mails, und für viele verschiedenen Plattformen (Android, iOS, Windows Phone, etc.).

Unabhängig vom Anwendungsgebiet und der Plattform, auf welcher eine App verfügbar ist, wird bei Apps zwischen 3 unterschiedlichen App-Typen unterschieden: Native Apps, Web Apps und Hybride Apps. Diese Unterscheidung bezieht sich auf die grundlegende technische Architektur, welche einer App zugrunde liegt.

3.1 Native App

3.1.1 Was ist eine Native App?

Ein Grossteil der über App-Stores erhältlichen Apps sind Native Apps. Wie das Wort „native“ (einheimisch) bereits sagt, handelt es sich bei Native Apps um Apps, welche bei einem bestimmten Betriebssystem „heimisch“ sind. Eine Native App wird spezifisch für ein bestimmtes mobiles Betriebssystem, z.B. für Android, entwickelt. Die App läuft dann nur auf dem Betriebssystem, für welches sie entwickelt wurde und muss zur Nutzung auf dem Smartphone installiert werden.

Möchte ich meine selbst entwickelte native App auch für ein anderes Betriebssystem verfügbar machen, so muss ich eine eigene Native App für das gewünschte Betriebssystem entwickeln.

Die Programmiersprachen zur Entwicklung von native Apps sind:

- Java, Kotlin, C++ für Android
- Swift und Objective-C für Apple iOS
- C#, C, C++ und JavaScript für Windows Phone



Abbildung 1: XelHa Native-App für Android

Native Apps haben folgende Vorteile:

- Sind spezifisch auf das jeweilige Endgerät respektive deren Plattform zugeschnitten. So können die spezifischen Guidelines der jeweiligen Plattform optimal eingehalten werden und eine intuitive Bedienbarkeit und ein einheitliches Look & Feel mit dem System gewährleistet werden.
- Libraries und Funktionen des jeweiligen Development Kit können verwendet werden. Bei Android können beispielsweise Java-Libraries verwendet werden.
- Können mit anderen, auf dem Endgerät installierten Apps, interagieren und auf diese zugreifen (z.B. auf Kamera-App). Voraussetzung ist, dass eine App so gebaut ist, dass andere Apps mit dieser interagieren können.
- Ermöglichen Systeminteraktionen, wie z.B. Push-Benachrichtigungen
- Können performant auf Hardwarefunktionen und Sensoren zugreifen und Daten auf dem Endgerät speichern.

Nachteile von Native Apps:

- Hoher Entwicklungs- und Wartungsaufwand. Für jedes Betriebssystem eine eigene Entwicklung.
- Vertrieb auf mehreren App-Stores

3.1.2 Entwicklung von Native Apps für Android

Android Studio ist die offizielle Entwicklungsumgebung (IDE) zur Entwicklung von Android Native Apps. Entwickelt wird hauptsächlich mit Java als Programmiersprache. Ausnahme sind GUI-Ansichten. Diese werden mit XML umgesetzt. Weitere Informationen zum Android Studio sind im Kapitel 2.2.1 zu finden.

Seit Android Studio 3.0 (Oktober 2017) kann anstelle von Java auch mit Kotlin entwickelt werden. Kotlin ist eine Programmiersprache, welche durch JetBrains vorangetrieben wird und 2016 erschien.

Im Rahmen dieses Moduls werden wir Android Studio mit Java noch im Detail kennen und anwenden lernen.

3.1.3 Entwicklung von Native Apps für iOS

iOS Native Apps können mit „Xcode“ der offiziellen Apple-Entwicklungsumgebung, entwickelt werden. Xcode ist nur für Mac OS X verfügbar, daher ist es nicht möglich, iOS-Apps auf einem Windows- oder Linux-PC zu entwickeln.

Die Apps für iOS werden mit den Programmiersprachen Swift oder Objective-C entwickelt. Swift ist eine Eigenentwicklung von Apple und wurde im September 2014 vorgestellt. Die Sprache ist objektorientiert und beinhaltet Ideen von Objective-C, Ruby, Python, C# und weiteren, eher unbekannten Programmiersprachen. Im Gegensatz zu Objective-C hat Swift die deutlich einfachere Syntax und ist näher an Programmiersprachen wie Java oder C#.

Objective-C ist der Vorgänger von Swift. Bis September 2014 konnten iOS-Apps nur mit Objective-C entwickelt werden. Seither können beide Programmiersprachen verwendet werden. Die Programmiersprachen sind untereinander kompatibel, d.h. Swift und Objective-C Code können gemixt werden. Somit müssen vor 2014 erstellte Apps zur Erweiterung nicht auf Swift umgeschrieben werden.

Nachfolgende Abbildung zeigt ein „Hello World“-Programm mit Swift und mit Objective-C.

Objective-c Hello World

```

1 #import <Foundation/Foundation.h>
2
3 @interface HelloWorld : NSObject
4 @end
5
6 @implementation HelloWorld
7
8 - (void)hello
9 {
10   printf("Hello World!");
11 }
12
13 @end
14
15 int main(int argc, char **argv)
16 {
17   id obj = [HelloWorld new];
18   [obj hello];
19   return 0;
20 }
```

Swift Hello World

```

1 import Foundation
2
3 class HelloWorld {
4   func hello() {
5     println("Hello World!");
6   }
7 }
8
9 let obj = HelloWorld()
10 obj.hello()
11 exit(0)
12
```

HelloWorld
HelloWorld

Abbildung 2 „Hello World“-Programm mit Swift und Objective-C [1]

Modul 335: Mobile-Applikationen realisieren

Die grössten Unterschiede zwischen der iOS-Entwicklung mit Xcode und Swift und der Android-Entwicklung mit Android-Studio und Java sind:

- **Simulator / Emulator:**

Xcode und auch Android-Studio bieten eine Möglichkeit, am PC zu testen, wie die App auf einem bestimmten Gerät aussieht. Bei Android ist dies der Emulator, bei iOS der Simulator. Der iOS-Simulator ist deutlich schneller als der Android-Emulator. Der Android-Emulator liefert jedoch deutlich realistischere Ergebnisse. Beim iOS-Simulator wird eine App häufig anders dargestellt als diese schlussendlich auf dem Gerät angezeigt wird.

- **App auf Gerät starten:**

Um eine iOS-App während der Entwicklung auf einem Gerät zu testen/starten, muss eine Apple-ID gelöst werden. Seit Xcode 7 (aktuell V9), kann diese zu Testzwecken kostenlos gelöst werden. Vor Xcode 7 war eine Mitgliedschaft beim Developer Program nötig, was jährlich 99\$ kostete. Bei Android kann die App ohne Account und irgendwelchen Kosten direkt auf dem Gerät gestartet/getestet werden.

- **Storyboard:**

Xcode bietet ein Storyboard an um die Ansichten miteinander zu verknüpfen und festzulegen, welche Ansicht welche andere Ansicht aufruft. Bei Android gibt's kein Storyboard, eine Ansicht muss im Java-Code eine andere Ansicht aufrufen.

- **Projektstruktur:**

Bei der Struktur des iOS-Projekts in XCode ist man relativ frei, Ordner und Dateien können erstellt werden wo man möchte und dann miteinander verknüpft werden. Bei Android hingegen ist die Projektstruktur relativ stark vorgegeben.

- **Lizenz für App-Store:**

Um eine Android-App im Google Play Store zu platzieren muss einmalig eine Developer Lizenz von 25\$ gelöst werden. Mit dieser Developer Lizenz können dann unbeschränkt viele Apps in den Google Play Store gestellt werden. Bei iOS muss eine Developer Lizenz von 99\$ pro Jahr gelöst werden um Apps in den Store zu stellen.

- **Verschiedene Gerätyphen:**

iOS-Apps laufen nur auf Apple Geräten. Die Anzahl verschiedenen Geräte und die unterschiedlichen Bildschirmgrößen sind daher beschränkt. Android läuft auf einer Vielzahl von Geräten unterschiedlicher Hersteller und mit unterschiedlicher Bildschirmgröße. Möglichst viele verschiedene Geräte zu unterstützen ist daher bei Android-Apps deutlich aufwendiger als bei iOS-Apps und führt zu einem grösseren Testaufwand.

Nachfolgende Abbildung zeigt das Storyboard von XCode.

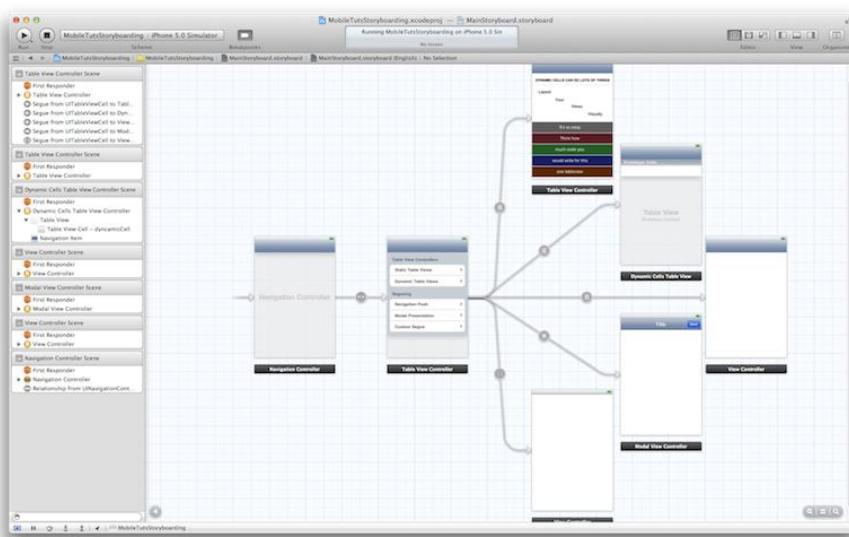


Abbildung 3 Storyboard von XCode [4]

3.2 Web App

3.2.1 Was ist eine Web App?

Eine Web App ist eine Webapplikation, welche über den Browser des mobilen Endgeräts aufgerufen wird. Eine Web App wird zwar im Browser des mobilen Endgeräts aufgerufen, verhält sich aber eher wie eine App als wie eine Webseite. Die Benutzeroberfläche ist optimiert für die Anzeige auf einem kleinen Bildschirm und die Elemente der Benutzeroberfläche sind näher den Elementen einer App als den Elementen einer klassischen Webseite. Das Stichwort hierzu ist „Responsive Webdesign“.

Viele grosse Firmen bieten von ihren Webapplikationen / Webseiten eine Web App an. Viele Webseiten leiten den Benutzer sofern er eine Webseite auf einem mobilen Endgerät aufruft, automatisch auf deren Web App um (sofern diese vorhanden ist).

Nachfolgende Abbildung zeigt die Web-App und die normale Webseite von immowelt.de und die Web App m.immowelt.de. Sie soll die grafischen Unterschiede veranschaulichen.

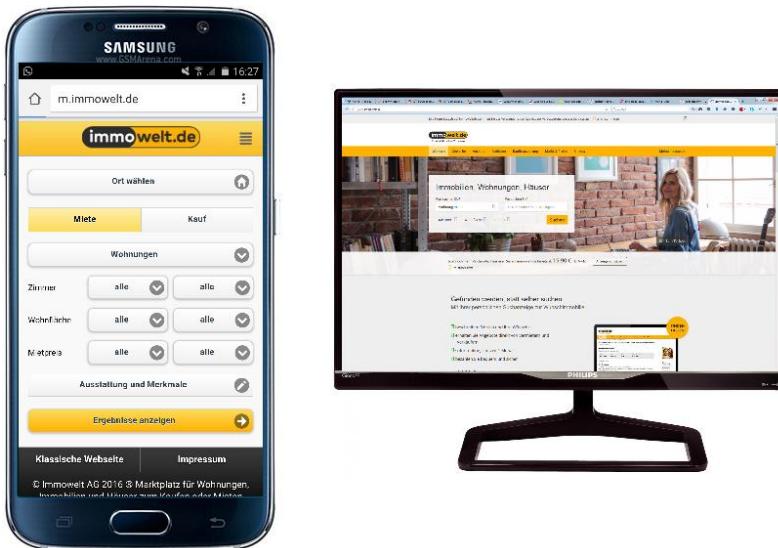


Abbildung 4 Unterschied Web App und normale Webseite am Beispiel von immowelt.de

Web Apps haben folgende Vorteile:

- Müssen nicht auf dem Endgerät installiert werden. Alles was der User zum Zugriff braucht ist ein Browser.
- Sind Plattformunabhängig. Eine Web App kann sowohl vom iOS, Android- oder Windows-Endgerät aufgerufen werden. Mit einer Entwicklung können also alle Plattformen abgedeckt werden.
- Updates bei der Web-App sind sofort für alle User verfügbar, da keine Installation.
- Werden von Suchmaschinen wie Google indexiert.

Nachteile von Web Apps:

- Sind meistens nur Online nutzbar. Ohne Internetverbindung kann die App auf dem mobilen Endgerät nur eingeschränkt genutzt werden.
- Hardwarefunktionen, Zugriff auf Sensoren und Systeminteraktionen sind nur sehr begrenzt möglich.
- Elemente der plattformspezifischen Software Development Kits können nicht genutzt werden.

3.2.2 Entwicklung von Web Apps

Web Apps werden mit HTML5, CSS3 und JavaScript entwickelt. Mit spezifischen CSS3-Templates ist es möglich, dass eine Web-App ähnlich aussieht wie eine Native Android- oder iOS-App.

Eine Web App kann ohne Probleme komplett von Hand erstellt werden. Bequemer geht es jedoch mit sogenannten Frameworks. Diese Frameworks bieten bereits fertige Komponenten, welche nur noch eingebunden werden müssen. So muss sich der Entwickler nicht mehr speziell um das Aussehen auf verschiedenen Plattformen kümmern, denn das Framework erledigt das für ihn.

Die Frameworks bestehen meistens aus JavaScript-Dateien, CSS-Dateien und Bildern, welche sich in der eigenen HTML5-Seite einbinden lassen. Dadurch können GUI-Elemente wie Buttons, Datepicker, Formulare, Listen, Navigationen, Checkboxen, Pop-ups, etc. mit auf mobile Geräte angepasstem Design und Funktionen erstellt werden.

Bekannte Frameworks zur Entwicklung von Web-Apps sind:

- **jQuery Mobile**
<http://jquerymobile.com>
Demo jQuery Mobile: <http://demos.jquerymobile.com/1.4.5/>
- **Bootstrap & Angular:**
<http://getbootstrap.com/>
<https://angular.io/>
- **Sencha Touch:**
<https://www.sencha.com/products/touch/>
- **Kendo UI:**
<http://www.telerik.com/kendo-ui>

3.3 Hybride App

3.3.1 Was ist eine hybride App?

Eine hybride App ist eine Mischung aus einer Native App und einer Web App. Das Ziel von hybriden Apps ist die Vorteile beider Architekturen zu vereinen und die Nachteile möglichst zu eliminieren.

Eine hybride App ist eigentlich eine native App, welche sobald diese gestartet wird, eine Web App anzeigt. Die Native App besteht mehr oder weniger nur aus einer Web Browser Komponente, einer sogenannten Web View. Die ganze Funktionalität der App wird als Web App umgesetzt und die Web App dann von der Native App in der Web View angezeigt. Der Benutzer lädt im jeweiligen App-Store die Native App, also den Rahmen der Hybriden App herunter.

Funktionen, wie Zugriff auf Sensoren oder Systeminteraktionen werden dann vom nativen Teil der App erledigt, die plattformunabhängigen Funktionen von der Web App.

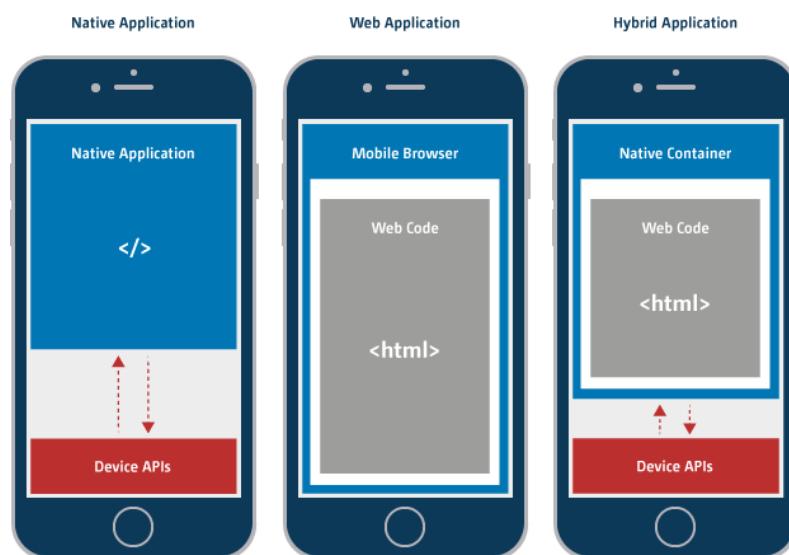


Abbildung 5 Vergleich des Aufbaus von Native Apps, Web Apps und Hybrid Apps [2]

Für die Entwicklung von Hybriden Apps existieren spezifische Entwicklungswerzeuge. Es ist nicht nötig mit Android Studio eine native App für Android zu erstellen, welche dann die Web App aufruft, mit XCode eine native App für iOS, welche dasselbe tut, etc. Mit den spezifischen Entwicklungswerzeugen kann der Code der hybriden App einmal geschrieben und dann für die verschiedenen Plattformen kompiliert werden.

Vorteile von Hybriden Apps:

- Verknüpfen diverse Vorteile von Native und Web Apps.

Nachteile von Hybriden Apps:

- Das GUI von hybriden Apps entspricht dem einer klassischen Web App. Plattformspezifische GUI-Elemente können auch bei hybriden Apps nicht benutzt werden, da im Endeffekt eine Web-App angezeigt wird. Mit sogenannten UI-Frameworks ist es jedoch möglich, nah an das Look & Feel einer Native App zu kommen.
- Performance ist schlechter als bei einer native App.

3.3.2 Entwicklung von hybriden Apps

Wie bereits im vorhergehenden Kapitel erwähnt existieren für die Entwicklung von Hybriden Apps spezifische Entwicklungswerkzeuge. Eine App muss nur einmal mit einem sogenannte hybriden Framework entwickelt werden und kann dann für die verschiedenen Plattformen (Android, iOS, Windows, etc.) generiert werden.

Der Web App Teil der hybriden App wird mit HTML5, CSS3 und JavaScript entwickelt. Für den Nativen Teil der App, also beispielsweise für den Zugriff auf die Hardware oder Sensoren, können sogenannte native JavaScript-APIs verwendet werden. Ein bekanntes Framework, welches native JavaScript-APIs zur Verfügung stellt ist Apache Cordova. Cordova ist in den meisten Hybrid-App-Frameworks enthalten und generiert aus dem Web App Code nativen Code für iOS-, Android- oder Windows-Plattformen.

Das bekannteste Framework zur Entwicklung von hybriden Apps ist Ionic. Ionic ist vollkommen Open Source und kann kostenlos heruntergeladen werden. Ionic verwendet für die Generierung des nativen Code Apache Cordova. Windows User mit installiertem Visual Studio Community Edition müssen Ionic nicht speziell installieren, sondern können die Visual Studio Tools für Apache Cordova verwenden und dann hybride Apps mit Visual Studio entwickeln.

Die Entwicklung einer hybriden App mit Ionic läuft wie folgt ab:

1. Mit HTML5, CSS3 und JavaScript wird die App entwickelt. Durch die Ionic CSS- und JavaScript-Klassen kann das GUI mobile-like gestaltet werden. Zudem beinhaltet Ionic das AngularJS-Framework. Dieses JavaScript-Framework erweitert das Vokabular von HTML und bietet viele sinnvolle Funktionen, wie beispielsweise Datenversionierung, ein Model-View-ViewModel-Muster und vieles mehr. Somit kann auch Angular-Code bei der Entwicklung der App eingesetzt werden.
2. Nachdem die App entwickelt wurde kann die App einerseits als Web App im Browser, andererseits als hybride App in iOS, Android und Windows-Simulatoren getestet werden. Diese Simulatoren sind Bestandteil von Ionic.
3. Zu guter Letzt kann die App in Ionic mit dem Cordova-Plugin für die verschiedenen Plattformen gebildet werden. Damit wird für jede Plattform eine native App erstellt, welche die Web App beinhaltet. Diese native App kann dann im entsprechenden App-Store hochgeladen oder auf dem Smartphone getestet werden.

Webseite von Ionic: <http://ionicframework.com>

Download von Visual Studio-Tools für Apache Cordova:

https://www.visualstudio.com/cordova-vs?wt.mc_id=o~display~ionic~dn948185

Weitere bekannte Frameworks zur Entwicklung von Hybrid Apps sind:

- **Adobe PhoneGap:**
<http://phonegap.com/>
- **Intel XDK:**
<https://software.intel.com/en-us/intel-xdk>
- **Onsen UI:**
<https://onsen.io/>
- **Famous:**
<http://famous.org/>
- **Framework 7:**
<http://framework7.io/#.VYM1T2AYz-N>

3.4 Native, Web und Hybrid App – Welcher App-Typ soll ich wählen?

Ob eine App als Native App, als hybride App oder als Web App entwickelt werden soll hängt schlussendlich von den Anforderungen des Kunden und den verfügbaren personellen und finanziellen Ressourcen ab.

Nachfolgende Abbildung vergleicht die 3 erläuterten App-Typen in Bezug auf mögliche Kundenanforderungen.

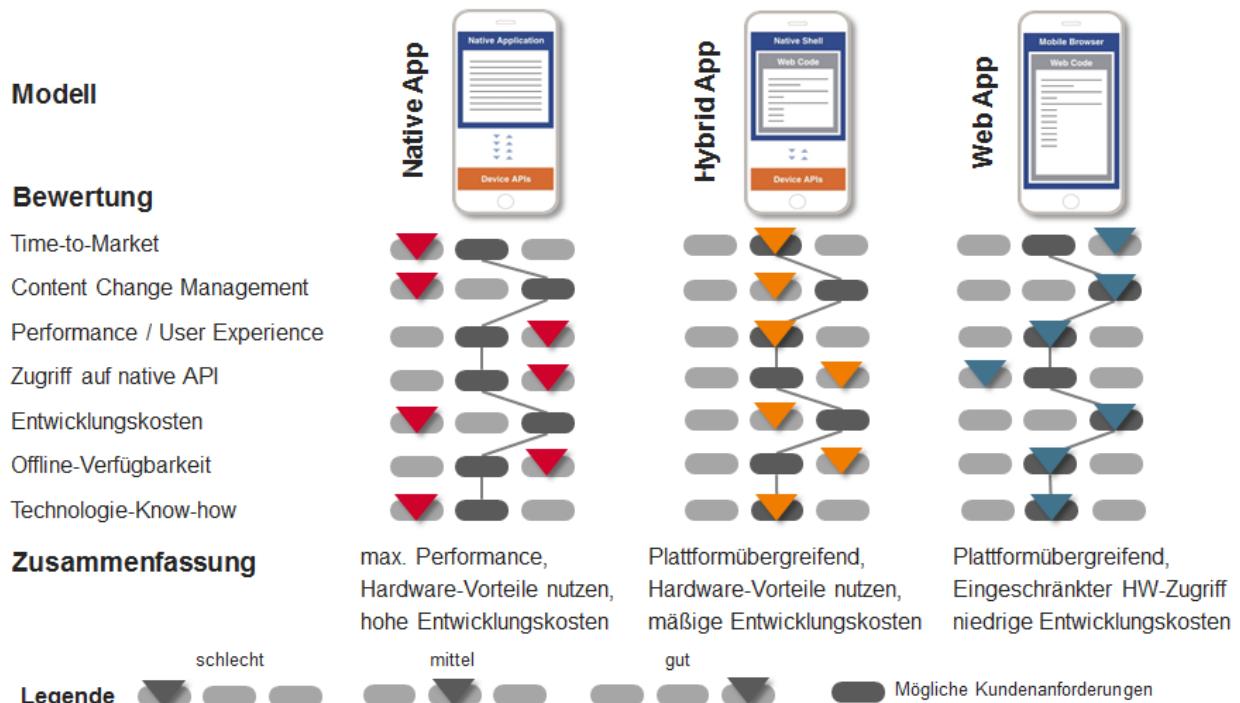


Abbildung 6: Vergleich von Native, Web und Hybrid App in Bezug auf mögliche Kundenanforderungen [31]

Eine native App soll gewählt werden, wenn die Performance, das Look & Feel des User Interfaces und die Nutzung von Hardwarefunktionen und Sensoren eine zentrale Rolle spielen. Dies ist beispielsweise bei Games, bei Social Networks oder bei Navigation-Apps der Fall.

Web Apps bieten sich an, wenn mit einer möglichst kurzen Entwicklungszeit alle Plattformen abgedeckt werden möchten und die Offline-Verfügbarkeit, das Plattformabhängige Look & Feel, sowie der Zugriff auf Sensoren nicht relevant sind. Viele Firmen entwickeln in einem ersten Schritt eine Web App um schnell am Markt zu sein, bevor dann später eine hybride oder Native App nachgeliefert wird.

Spielen die bei der Native App genannten Kriterien eine weniger zentrale Rolle, so ist eine hybride App die beste Lösung. Dies ist vor allem bei Business-, Utility- und Finanz-Apps der Fall. Diese Gruppe von Apps macht einen Grossteil der Apps im App Store aus. In den letzten Jahren haben sich die Frameworks zur Entwicklung von hybriden Apps massiv verbessert und die Nachteile von hybriden Apps gegenüber Native Apps wurden erheblich reduziert.

3.5 Cross Plattform Entwicklung

Mit einem Cross Plattform Framework können mit einer Entwicklungsumgebung Apps für verschiedene Plattformen (Android, iOS, Windows Mobile) entwickelt werden. Es handelt sich jedoch bei den entwickelten Apps nicht um hybride Apps, sondern um native Apps.

Das funktioniert so, dass die Businesslogik von der Darstellung der App getrennt wird. Die Darstellungsschicht und auch die plattformspezifischen Funktionen werden einzeln für jede Plattform entwickelt. Die Businesslogik wird in einem eigenen Projekt umgesetzt und dieses Projekt kann dann in die plattformspezifischen Apps als Library eingebunden werden.

Beim Komplizieren der App für die jeweiligen Plattformen wird dann die Businesslogik in jede App hineingeplant. Es handelt sich damit um eine komplette Native App und nicht um eine Hybrid App.

Das bekannteste Cross Plattform Framework ist Xamarin. Seit Februar 2016 gehört Xamarin zu Microsoft. Entwickelt wird entweder mit dem Xamarin Studio oder mit Visual Studio in der Programmiersprache C#. Xamarin stellt 100% der nativen APIs für iOS, Android und Windows zur Verfügung. Damit können mit Xamarin alle plattformspezifischen Funktionen optimal genutzt werden.

Neben der Trennung von Businesslogik und Darstellungsschicht gibt es noch eine zweite Art, Apps mit Xamarin zu entwickeln. Und zwar kann das GUI mit Xamarin.Forms erstellt werden. Damit kann im Idealfall nahezu 100% des Codes für jede mobile Plattform verwendet werden. Das GUI muss also mit Xamarin.Forms nicht mehr einzeln für jede Technologie entwickelt werden.

Nachfolgende Abbildung zeigt, wie eine Xamarin App mit Businesslogik (Backend) und User Interface strukturiert sein kann.

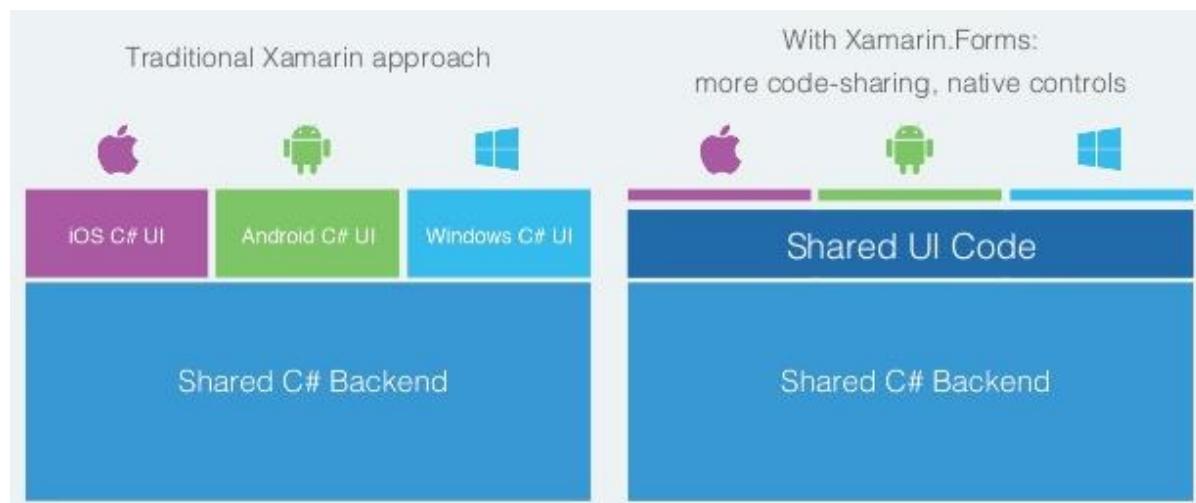


Abbildung 7 Cross Plattform Entwicklung mit Xamarin [5]

Xamarin ist an die Lizenz von Visual Studio gebunden. Visual Studio ist für Studenten kostenlos verfügbar, für Firmen fallen jedoch ziemlich schnell hohe Kosten an.

Webseite von Xamarin: <https://www.xamarin.com/>

4 Android

Das vorhergehende Kapitel zeigte die verschiedenen App-Typen auf. Die folgenden Kapitel dieses Dokuments beziehen sich nun auf den Kernteil des ÜK Modul 335 und zwar auf die Entwicklung einer Native App für Android. Android ist das weltweit am weitesten verbreitete Betriebssystem für mobile Endgeräte, Tendenz eher steigend.

Die nachfolgende Statistik zeigt den Marktanteil der verschiedenen Betriebssysteme auf mobilen Endgeräten im Jahr 2017 in der Schweiz

Marktanteil mobile Betriebssysteme CH 2017

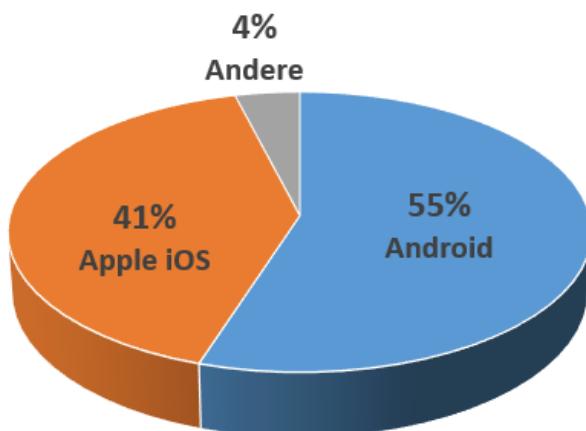


Abbildung 8 Marktanteil Mobile Betriebssysteme in der Schweiz [6]

In ganz Europa gesehen ist Android der absolute Marktführer.

Marktanteil mobile Betriebssysteme Europa

2017

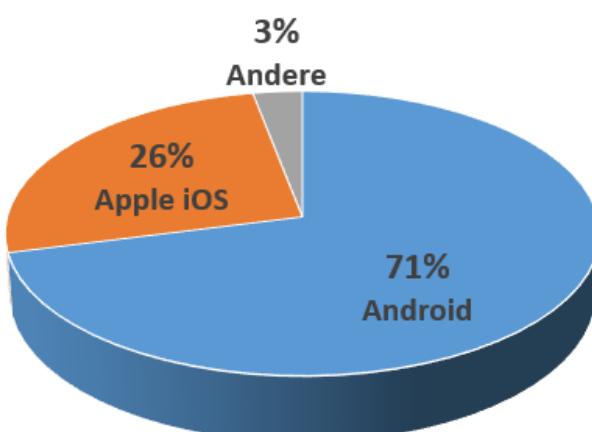


Abbildung 9 Marktanteil Mobile Betriebssysteme in Europa [7]

Nachfolgendes Kapitel gibt einen Einblick in die historische Entwicklung von Android und vermittelt grundlegende Informationen zur Systemarchitektur und dem Aufbau der Android-Plattform zur Entwicklung von Android-Apps.

4.1 Entstehung und historische Entwicklung

Die Firma Android wurde bereits 2003 gegründet. Damals war Android ausschliesslich zur Steuerung von Digitalkameras gedacht. Im Sommer 2005 wurde Android von Google aufgekauft. Google änderte schlagartig die Pläne und gab im November 2007 bekannt, ein Betriebssystem unter dem Namen „Android“ für Mobiltelefone zu entwickeln.

Im Oktober 2008 wurde Android in der Version 1.0 offiziell vorgestellt. Über die Jahre wurde Android dann stetig weiterentwickelt. Aktuell sind wir bei Version 8.1 (Stand Januar 2018).

Im Folgenden werden die Android-Versionen mit den wichtigsten Änderungen aufgeführt.

1.5 (Cupcake, 30.4.2009):

Android unterstützt nun Geräte ohne Hardwaredatatur.

2.0 (Eclair, 3.12.2009):

Komplett neue Programmierschnittstelle, welche den Zugriff auf Kontaktdaten ermöglicht.

2.3 (Gingerbread, 6.12.2010):

Sensoren werden nun unterstützt (Barometer, Gyroskope, etc.)

3.x (Honeycomb, ab 23.2.2011):

Neue benutzerfreundlichere Oberfläche. Ansichten für Tablets wurden optimiert.

4.0 (Ice Cream Sandwich, ab 19.10.2011):

Ab dieser Version stand der Quellcode von Android anderen Mobiltelefon-Herstellern (z.B. Samsung) zur Verfügung. Zugriff auf Kalenderfunktionen ist möglich.

4.1 – 4.3 (Jelly Bean, ab 27.6.2012):

Detailverbesserungen an der Darstellung, Geschwindigkeitsoptimierungen. OpenGL 3.0 wird unterstützt.

5.x (Lollipop, ab 3.11.2014):

Das Design wurde grundlegend überarbeitet. Es gibt keine Trennung mehr zwischen Smartphone, Tablet und Wearables (Uhren, etc.). Zudem wurde die virtuelle Maschine „Dalvik“ durch ART (Android Runtime) ersetzt. Weiter werden nun 64-Bit Prozessoren unterstützt.

6.x (Marschmallow, 5.10.2015):

Ein neues Berechtigungssystem wurde eingeführt. Berechtigungen für Apps können eingeschränkt werden. Auch möglich ist es nun, mehrere Apps gleichzeitig auf einem Bildschirm anzuzeigen (Multi-Fenstermodus). Zudem wird ab Android 6.0 der Fingerabdruckscanner unterstützt.

7.x (Nougat, 22.08.2016):

Viele Funktionen (Benachrichtigungen, Einstellungen, Multitasking, Split Screen) wurden verbessert. Zudem werden ein VR-Modus und das Google VR SDK unterstützt.

8.x (Oreo, 21.08.2017):

Komplett überarbeitete Einstellungen, optimierter Dateimanager, längere Akkulaufzeit, Überprüfung von Apps auf schadhaftes Verhalten, und viele weitere Optimierungen.

Die Aktuelle Version 8.1 wurde am 6. Dezember 2017 herausgegeben.

Eine detaillierte Auflistung aller Android-Versionen und deren wesentliche Neuerungen sind auf folgender Seite zu finden: https://de.wikipedia.org/wiki/Liste_von_Android-Versionen

4.2 Systemarchitektur

Das Android-Betriebssystem basiert auf einem Linux Kernel. Dieser verwaltet die Prozesse und den Speicher und stellt Treiber für die Hardware zur Verfügung.

Nachfolgende Abbildung zeigt die Systemarchitektur der Android-Plattform. Diese ist in 5 (6) Schichten unterteilt, welche in den nachfolgenden Kapiteln detaillierter betrachtet werden.

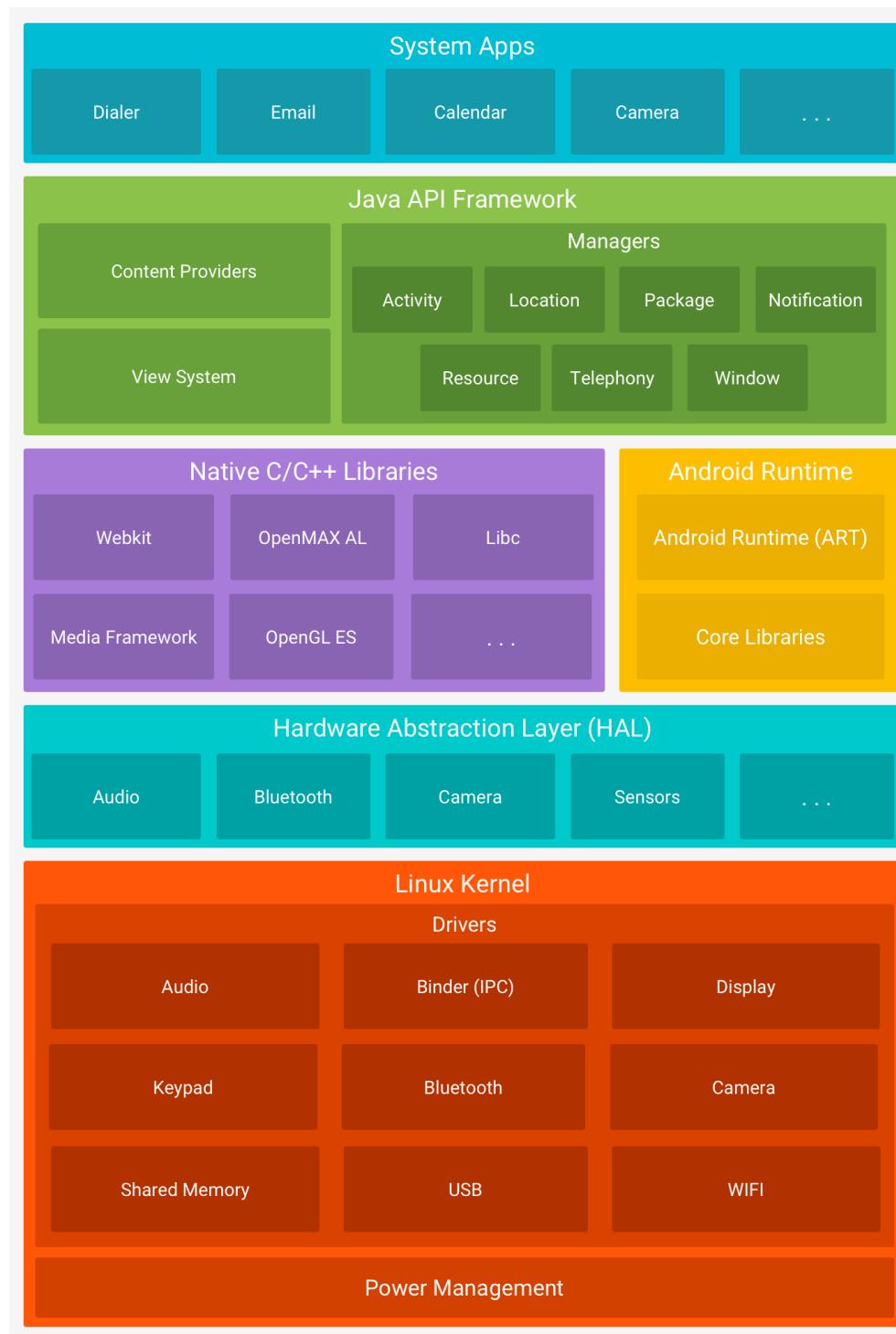


Abbildung 10 Systemarchitektur der Android Plattform [8]

4.2.1 Android Runtime

Bis Android 4.4 wurden Android-Apps von der virtuellen Maschine „Dalvik“ ausgeführt. Seit Android 5.0 wurde Dalvik von der Android Runtime, kurz ART, abgelöst.

Dalvik wandelte den Java-Bytecode einer App in den Befehlssatz des Prozessors um. Der umgewandelte Code wurde dann auf dem mobilen Gerät ausgeführt. Dalvik wendete eine Just-in-time-Kompilierung an, d.h. bei jedem Aufruf der App wurde der Bytecode zunächst umgewandelt. Dies führte zu einer verzögerten Ausführung der Apps.

ART hingegen wandelt den Bytecode nicht bei jedem Start der App um, sondern nur einmal und zwar bei der Installation der App. Dadurch fällt die Just-in-time-Kompilierung beim Start der App weg, was einerseits die Startgeschwindigkeit einer App verbessert, andererseits zu einem geringeren Energieverbrauch führt, weil nicht bei jedem App-Start Akkuleistung für die Kompilierung verschwendet wird.

4.2.2 Hardware Abstraction Layer

Der Hardware Abstraction Layer stellt Schnittstellen zur Verfügung, welche Hardwarefunktionen für das Java-API-Framework nutzbar machen. Jedes Modul in diesem Layer ist für eine Hardwarefunktion zuständig, beispielsweise für die Kamera, für die Sensoren, für das Bluetooth-Modul, etc. Dank diesem Layer kann ich mit dem Android-Framework und Java auf Gerätehardware zugreifen.

4.2.3 Libraries

Bei den Libraries handelt es sich um Libraries, welche der App-Schicht Dienste zur Verfügung stellen. Diese nativen Libraries sind in C oder C++ geschrieben.

Die Library SQLite beispielsweise ermöglicht einer App, auf SQLite-Datenbanken zuzugreifen. OpenGL/ES wird für komplexere Grafikprogrammierungen, u.a. für Games benötigt. OpenSSL für den Zugriff von https-Webseiten, usw.

4.2.4 Java API Framework

Das Java API Framework vereinfacht die Erstellung eigener Apps. Es stellt, wie jedes Framework, Klassen zur Verfügung, welche bestimmte Aufgaben in der App erledigen können.

Es stehen Klassen zur Gestaltung von Ansichten, Klassen zum Zugriff auf Gerätehardware oder Sensoren, Zugriff auf Kontakte oder Dateien, Zugriff auf GPS und andere Lokalitätsdienste und noch viel mehr zur Verfügung.

Die wichtigsten Teile des Application Frameworks sind:

- **View System (Views):** Views sind GUI-Komponenten wie z.B. Textfelder, Dropdowns, Checkboxen, Buttons, etc.
- **Activity-Manager:** Steuert den Lebenszyklus einer Anwendung. Dazu gehört beispielsweise der Start, das Minimieren/Maximieren einer App oder das Beenden einer App. Weitere Informationen zum Activity Lifecycle folgen später in diesem Dokument.
- **Notification Manager:** Ermöglicht Zugriff auf die Android-Statuszeile. Damit können Push-Nachrichten, wie man sie beispielsweise von Whats-up oder Facebook kennt, erstellt werden.
- **Resource Manager:** Zugriff auf Dateien (z.B. Grafiken, Textfiles, etc.)

4.2.5 System Apps

Android ist nicht nur ein Betriebssystem, sondern beinhaltet eine Reihe von vorinstallierten Apps, welche mehr oder weniger zum Betriebssystem dazugehören. Dies ist beispielsweise die App zum Telefonieren, der vorinstallierte Browser, der Home-Bildschirm oder die Kontaktliste. Daneben gibt es noch viele weitere vorinstallierte Apps, wie z.B. Google Play, etc.

4.3 Android SDK

Das Android Software Development Kit (SDK) ist eine Sammlung von Programmen / Komponenten um Android-Apps zu erstellen. Um Apps mit Android Studio zu entwickeln, muss mindestens ein Android SDK installiert sein. Mit der Installation von Android Studio wird per Default ein SDK installiert.

Für jede Android Version wird ein SDK herausgegeben, welches die Android-Plattformen und Komponenten der jeweiligen Android-Version beinhaltet. Möchte ich eine App entwickeln, welche Komponenten beinhaltet, die erst mit Android 8 neu hinzugekommen sind, so muss ich das SDK von Android Version 8 zwingend installiert haben.

Erstellt man in Android Studio ein neues Projekt, so kann das Minimale SDK, d.h. die minimale Android-Version, auf welcher die App laufen soll, angegeben werden. Basierend auf diesem SDK wird die App dann gebaut. Heute (Januar 2018) verwenden 100% der weltweit vorhandenen Android-Geräte Android 4.0.3 (Ice Cream Sandwich) oder neuer. Darum macht es Sinn, Android 4.0.3 als minimales SDK zu verwenden. Dieses SDK muss dann in Android Studio installiert sein, damit die App programmiert werden kann.

4.3.1 Android SDKs installieren

Android SDKs können in Android Studio über das Menü „Tools → Android → SDK Manager“ installiert werden.

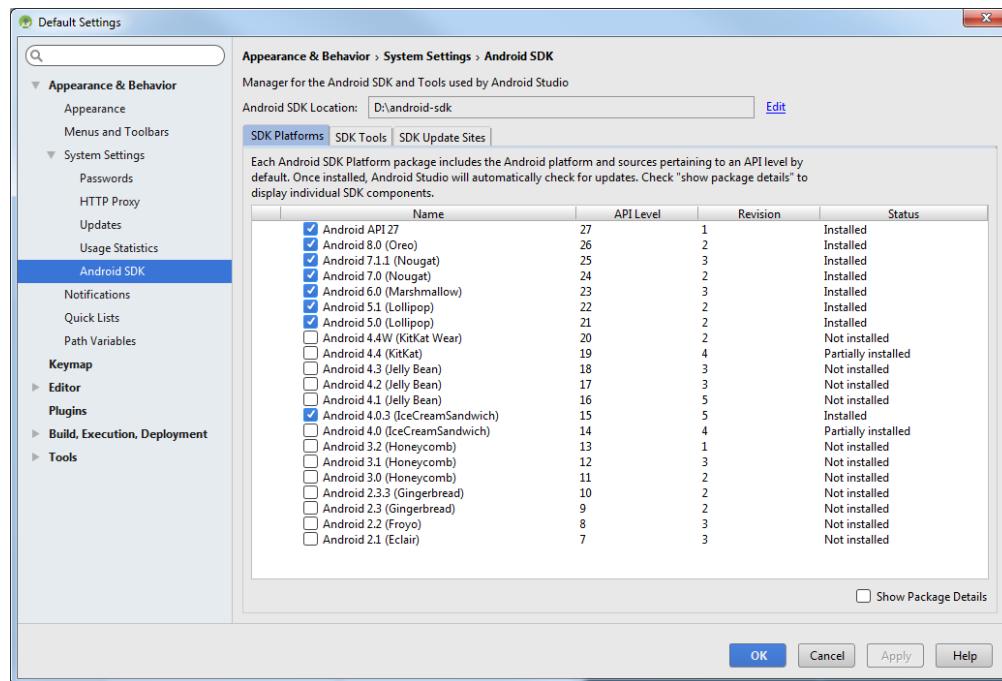


Abbildung 11 Android SDKs installieren

Danach den Haken bei der gewünschten Version aktivieren und den Button „Apply“ drücken. Nun müssen Sie noch die Lizenzbestimmungen akzeptieren und schon wird das SDK installiert.

4.4 Android Virtual Device Manager

Im Android Device Manager (kurz AVD Manager) können die virtuellen Geräte verwaltet werden, d.h. die Geräte, welche später ein Emulator simuliert werden möchten.

Ein Emulator bietet die Möglichkeit, in einem Fenster auf dem PC ein Smartphone, ein Tablet, ein Wearable oder ein TV darzustellen / zu simulieren und die erstellte App in diesem Emulator zu testen. Dies macht Sinn, wenn ich als Entwickler schauen möchte, wie sich meine App auf einem bestimmten Gerät verhält, ich das Gerät jedoch physisch nicht besitze.

Jedes virtuelle Gerät, welches im AVD Manager erstellt wird, kann im Emulator simuliert werden. Möchte die eigene App beispielsweise auf einem Samsung Galaxy S5, Galaxy S6, S7 und S8 getestet werden, können im AVD Manager all diese virtuellen Geräte erstellt und die App dann im Emulator für all diese Geräte getestet werden.

Ein virtuelles Gerät wird immer für ein vordefiniertes Hardware-Profil erstellt. Neben dem Hardware-Profil des Geräts muss zudem die Android-Version, welche emuliert wird, angegeben werden. Android Studio verfügt per Default über Hardware-Profile für alle Nexus-Geräte. Möchte ich ein anderes Gerät emulieren, wie beispielsweise die genannten Galaxy Smartphones, so muss ich ein eigenes Hardware-Profil erstellen.

Ein Hardware-Profil charakterisiert sich durch die Bildschirmgrösse, die Grösse des RAMs, die unterstützten Ausrichtungen, die vorhandenen Kameras und die vorhandenen Sensoren. Durch diese Charaktereigenschaften kann jedes Android-Gerät simuliert werden. Es ist also grundsätzlich möglich, die eigene App für jedes Android-Gerät und jede Android-Version zu testen, ohne dass ich das Gerät tatsächlich besitze. Natürlich macht dies bei der ungeheuren Anzahl an unterschiedlichen Android-Geräten keinen Sinn. Wie eine App im Emulator optimal getestet wird, wird im Kapitel 12.2.3 erläutert.

4.4.1 AVD Manager starten

Der AVD Manager kann in Android Studio über das Menü „Tools → Android → AVD Manager“ gestartet werden. Angezeigt werden dann in einer Liste alle erstellten virtuellen Geräte.

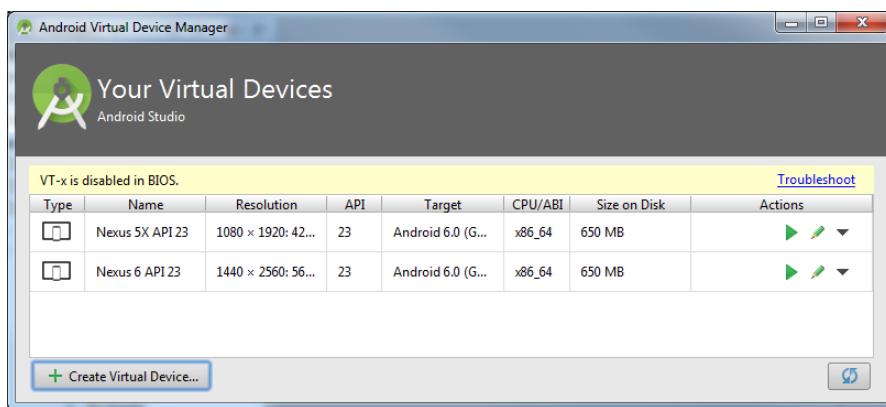


Abbildung 12 AVD Manager

4.4.2 Neues virtuelles Gerät für vordefiniertes HW-Profil erstellen

1. Klicken Sie auf den Button „**Create Virtual Device..**“. Danach können Sie ein vordefiniertes Hardwareprofil auswählen oder ein neues Hardwareprofil erstellen. Im nachfolgenden Beispiel wird ein virtuelles Nexus 5x erstellt. Dabei handelt es sich um ein bereits vordefiniertes Profil.
2. Gewünschtes **Hardware-Profil (Nexus 5x)** auswählen. Button „**Next**“ klicken.

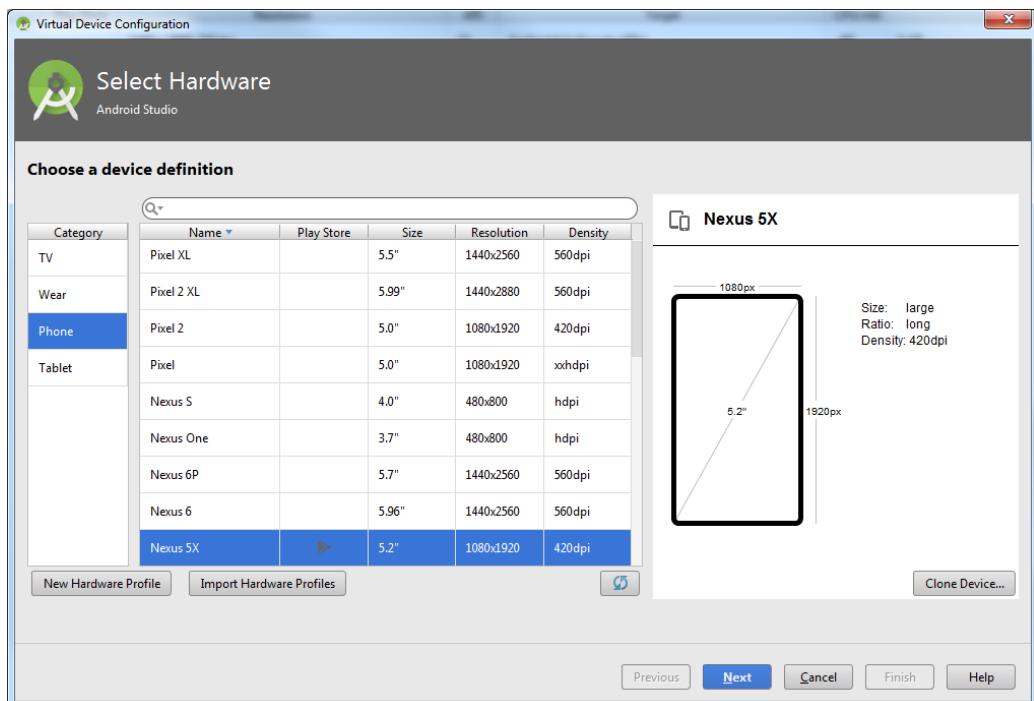


Abbildung 13 Neues virtuelles Gerät erstellen (Hardware Profil auswählen)

3. Das **Betriebssystem auswählen**, welches emuliert werden soll (z.B. Marshmallow). Die gängigen Betriebssysteme für das ausgewählte Gerät sind im Tab „Recommended“ zu finden. Wird eine andere Android-Version gewünscht, lässt sich diese im Tab „Other Images“ finden. Möglicherweise müssen Sie noch das Image der Android-Version, welche Sie auf dem Smartphone installieren möchten, herunterladen. Klicken Sie dazu einfach auf den Download-Link hinter der Android-Version.

[Oreo Download](#)

4. Button „**Next**“ klicken
5. Nun können Sie den **Namen** angeben, unter welchem das virtuelle Gerät im AVD Manager auftauchen soll, wie auch Angaben zur Default-Ausrichtung des Geräts. Die Default-Einstellungen sind meistens OK.
6. Button „**Finish**“ klicken. Das virtuelle Gerät wird nun erstellt.

4.4.3 Neues virtuelles Gerät mit eigenem HW-Profil erstellen

Mit nachfolgenden Schritten können Sie ein virtuelles Samsung Galaxy S6 erstellen:

1. Klicken Sie auf den Button „**Create Virtual Device..**“ im AVD Manager.
2. Klicken Sie auf den Button „**New Hardware Profil**“.
3. Erstellen Sie nun das Hardware-Profil mit den Angaben des Samsung Galaxy S6. Die benötigten Angaben finden Sie für praktisch jedes Smartphone im Internet.

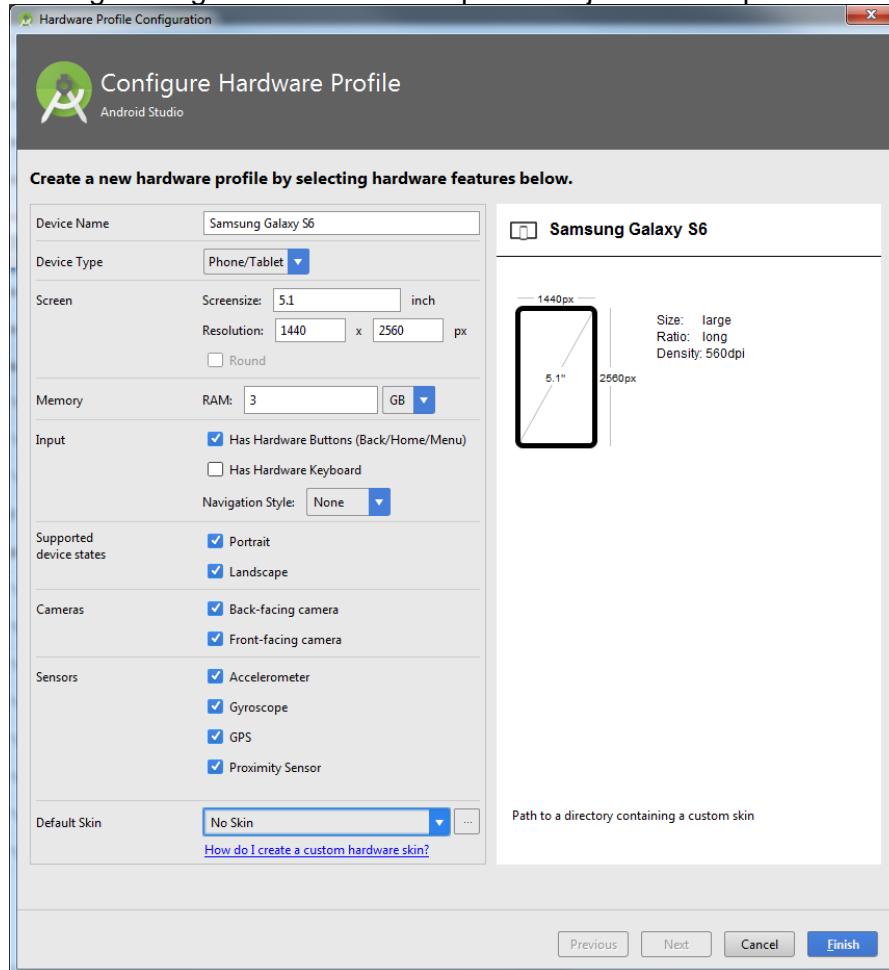


Abbildung 14 Hardware-Profil für Samsung Galaxy S6 erstellen

4. Button „**Finish**“ klicken. Das Hardware-Profil wird nun erstellt und kann dann in der Liste der Hardware-Profile ausgewählt werden.
5. Um das virtuelle Gerät für dieses Hardware-Profil zu erstellen, müssen Sie nun das Profil in der Liste auswählen und die Schritte 2-6 des Kapitels 4.4.2 durchführen.

4.4.4 Virtuelles Gerät (Emulator) starten

Sie können das virtuelle Gerät über den **Play-Button** (▶) im AVD Manager starten.

4.5 Gradle als Build Management Tool

Android Studio verwendet „Gradle“ um den Build einer App zu erstellen. Die Gradle-Files mit der Endung .gradle befinden sich in einem Android-Projekt und können direkt in Android-Studio bearbeitet werden. In diesen Gradle-Files kann der Entwickler Konfigurationen für den Build der App vornehmen. Das zentrale Gradle-File für die Konfiguration ist die Datei „build.gradle“.

Nachfolgend ein paar Konfigurationen, welche im build.gradle definiert werden:

- Android SDK-Version, für welche die App kompiliert werden soll.
- Eindeutige ID der App, z.B. „ch.nyp.xelha“. Muss eindeutig sein für eine App innerhalb des ganzen Google Play Stores.
- Versionsnummer und Versionsbezeichnung der App
- Abhängigkeiten der App (Importierte Libraries)

Die folgende Abbildung zeigt das build.gradle der XelHa-App.

```
apply plugin: 'com.android.application'

android {
    signingConfigs {
    }
    compileSdkVersion 22
    buildToolsVersion "21.1.2"
    defaultConfig {
        applicationId "ch.nyp.xelha"
        minSdkVersion 13
        targetSdkVersion 22
        versionCode 10
        versionName "1.16.13"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    productFlavors {
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile 'com.android.support:appcompat-v7:22.1.1'
    compile 'com.android.support:support-v4:22.1.1'
    compile project(':android-support-v4-preferencefragment')
    compile project(':qrcodescanner')
    compile 'com.squareup.retrofit:retrofit:1.6.1'
    compile 'com.google.code.gson:gson:2.3'
    compile 'org.parceler:parceler:0.2.13'
    compile 'com.j256.ormlite:ormlite-core:4.48'
    compile 'com.j256.ormlite:ormlite-android:4.48'
    compile files('libs/logback-android-1.1.4.jar')
    compile files('libs/slf4j-api-1.7.6.jar')
}
```

Abbildung 15 build.gradle der XelHa-App

Als Entwickler hat man meistens mit der Erstellung des Inhalts der verschiedenen Gradle-Files nicht viel zu tun. Die Angaben wie SDK-Versionen oder Application-ID können bei der Erstellung einer neuen App über das GUI angegeben werden. Android Studio übernimmt diese Angaben dann automatisch ins Gradle-File. Werden jedoch Third Party Libraries oder Proguard verwendet, kommt man um eine manuelle Erweiterung des build.gradle nicht herum.

5 Das erste eigene Android-Projekt

Dieses Kapitel zeigt Schritt für Schritt, wie Sie mit Android Studio eine einfache „Hello World“-App erstellen können. Die App zeigt in der App Bar (auch als Action Bar bekannt) und im Content-Bereich den Text „HelloWorld“ an.



Abbildung 16 HelloWorld-App

5.1 Projekt erstellen

Am Anfang muss in Android Studio ein neues Projekt für die Entwicklung der App erstellt werden. Gehen Sie dazu wie folgt vor:

1. Klicken Sie das Menü „File → New → New Project“.
2. Nun öffnet sich ein Assistent, welcher Sie durch die Erstellung des Projekts führt.
3. Machen Sie nun im ersten Schritt **folgende Angaben** und klicken Sie danach auf den Button „Next“:

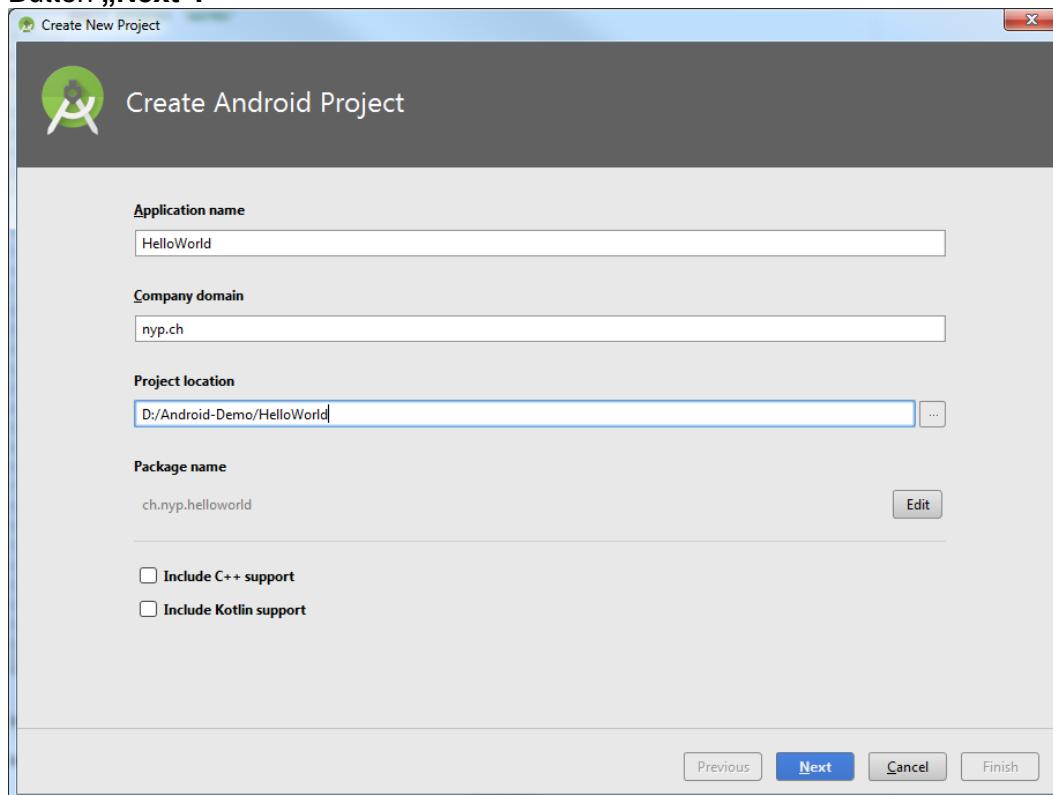


Abbildung 17 Neues Projekt erstellen – Projektname angeben

Application name: Name der App. Unter diesem Name wird die App auf dem Smartphone angezeigt.

Company Domain: Domain der Firma für welche die App entwickelt wird, oder die Domain eines selbständigen Entwicklers. Diese Domain wird für die Generierung des Package-Name benötigt (siehe ausgegraute Angabe). Der Package Name muss eindeutig sein für jede App im Google Play Store. Da Domains weltweit eindeutig sind, wird dieser aus der Domain generiert.

Project location: Pfad wo das Projekt auf der Festplatte gespeichert werden soll.

4. Wählen Sie nun, für welche Plattform Sie eine App entwickeln möchten. Wir konzentrieren uns in diesem ÜK auf die Erstellung von Apps für Smartphone/Tablets.
 Machen Sie daher den Hacken bei „**Phone und Tablet**“. Wählen Sie zudem die tiefste Android-Version aus, auf welchem Ihre App laufen soll.

Empfehlung: **Android 4.0.3**, da 100% aller Geräte, welche im Google Play Store aktiv sind, Android 4.0.3 oder höher verwenden. Dem Entwickler stehen bei der Entwicklung nur die Komponenten zur Verfügung, welche bis zum gewählten Minimum-SDK integriert wurden.

- Klicken Sie danach auf den Button „**Next**“.
5. Nun können Sie zu ihrer App eine Activity hinzufügen. Eine Activity kann im Moment als eine Ansicht betrachtet werden (weitere Infos folgen). Damit kann also gewählt werden, welche Art von Ansicht beim App-Start angezeigt werden soll.

Wählen Sie „**Empty Activity**“ und klicken Sie den Button „**Next**“. Damit wird eine nahezu leere Ansicht erstellt, was meistens die richtige Wahl ist. Bei den anderen Activities wird bereits recht viel Code erstellt, beispielsweise ein Login-Dialog oder eine Google Maps Ansicht.

6. Geben Sie nun den Namen des Activity und den Layout-Name ein. Eine Activity besteht immer aus einer Java-Klasse und aus einem XML-File für das Look & Feel. Jede Activity-Java-Klasse soll **mit „Activity“ enden**. Das dazugehörige XML-File mit **„activity“ beginnen**. Bei Java-Klassen hat sich die Upper CamelCase als Schreibweise eingebürgert. Bei XML-Files hingegen wird immer alles klein geschrieben und mehrere Wörter durch ein Underline (_) voneinander getrennt.

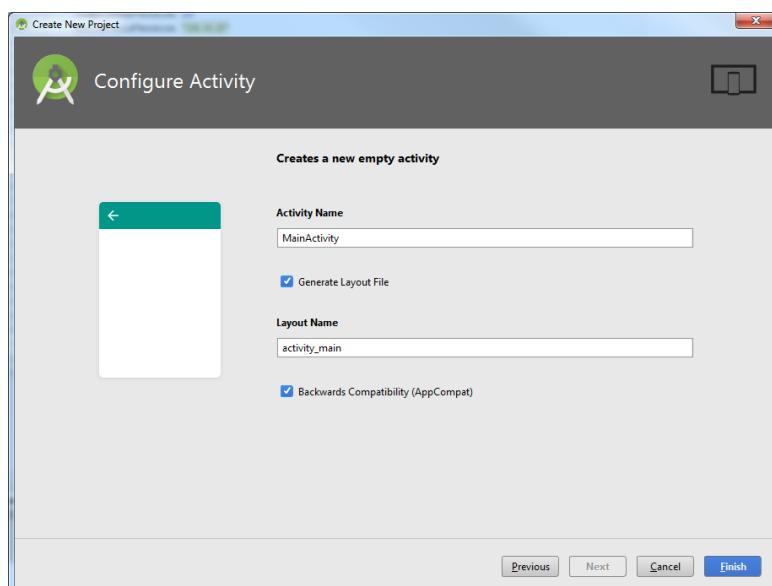


Abbildung 18 Neues Projekt erstellen - Activity Name und Layout Name

7. Schliessen Sie die Projekterstellung mit dem Button „**Finish**“ ab. Android Studio erstellt nun bereits automatisch eine „HelloWorld“-App, weil wir bei Schritt 5 „Empty Activity“ ausgewählt haben. Um eine vollständig leere App zu erzeugen, müsste bei Schritt 5 „No Activity“ ausgewählt werden.

Im Kapitel 5.2 betrachten wir nun die durch Android Studio erstellten Dateien und Verzeichnisse.

5.2 Projektstruktur

Nachdem Android-Studio das Projekt angelegt hat sehen Sie Links in Android Studio unter „Project“ die erstellten Verzeichnisse, Packages und Dateien.

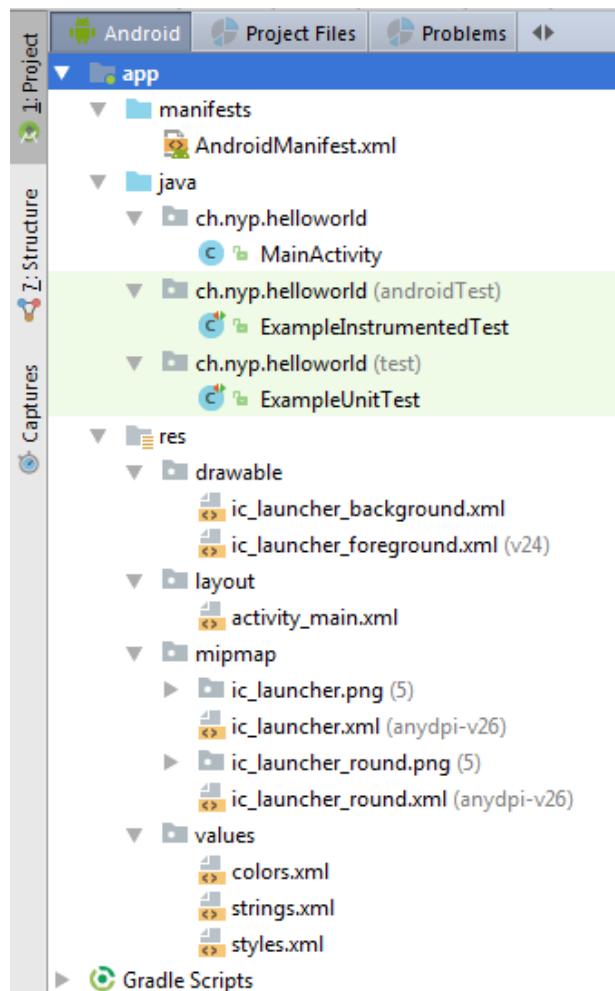


Abbildung 19 Projektstruktur „HelloWorld“-App

5.2.1 Java-Verzeichnis

In diesem Verzeichnis befinden sich alle Java-Klassen, d.h. der eigentliche Programmcode der App.

Package ch.nyp.helloworld:

Im Java-Verzeichnis wurde das Root-Package „ch.nyp.helloworld“ erstellt, welches wir bei Schritt 3 der Projekterstellung angegeben haben. Innerhalb dieses Packages werden bei einer App alle weiteren Java-Klassen hinzugefügt, d.h. Klassen für die Logik der App, die Model-Klassen, die Klassen für das Event-Handling, Klassen für den Datenbankzugriff, etc. Natürlich sollten innerhalb dieses Packages weitere Packages erstellt werden um die Java-Klassen zu gliedern.

In unserem HelloWorld-Projekt wurde die Klasse „MainActivity“ erstellt. Diese Klasse wird beim App-Start aufgerufen. Sie macht im Moment nicht mehr, als das GUI (XML-File activity_main) darzustellen. Jede GUI-Ansicht (Activity oder Fragment) besteht aus dem XML-File für die Anzeige des GUIs (siehe Ressourcen) und einem dazugehörigen Java-File, welches die Daten für diese Ansicht aufbereitet und Ereignisse entgegennimmt oder auslöst (z.B. Button-Klick).

Test-Packages ch.nyp.helloworld

Neben dem Package für die Java-Klassen der App wurden zwei Packages für Whitebox-Tests (Unit Tests) erstellt. Weitere Infos zu Unit Tests folgen im Kapitel 12.1.

5.2.2 Ressourcen (res)

Im res-Verzeichnis werden die sogenannten Ressourcen hinterlegt. Dies sind XML-Files oder Bilder, welche für die Darstellung des GUIs verantwortlich sind. Das GUI einer App wird in diesen XML-Files programmiert. Die Java-Klassen sind dann verantwortlich dafür, die GUI-Elemente der XML-Files mit Daten abzufüllen oder auf die Ereignisse des GUIs (z.B. Button-Klick) zu reagieren.

Das res-Verzeichnis seinerseits ist weiter unterteilt in folgende 4 Verzeichnisse:

layout

Im Layout-Verzeichnis werden die GUI-XML-Files der Ansichten hinterlegt. Diese werden auch Layout-Dateien genannt. Dazu gehören die XML-Files von Activities oder Fragments, layouts für Spinners (Dropdowns), etc. Weitere Infos hierzu erhalten Sie im weiteren Verlauf dieses Dokuments.

Bei der HelloWorld-App befindet sich in diesem Verzeichnis die Datei „activity_main.xml“. Dies ist die GUI-Ansicht, die beim App-Start aufgerufen wird und den Text „Hello World!“ ausgibt. Wenn Sie auf die Datei „activity_main.xml“ doppelklicken, öffnet sich die Ansicht im GUI-Designer. Hier kann die GUI-Ansicht von Hand zusammengeklickt werden. Wechseln Sie unten Links in das Tab „Text“, so erscheint der XML-Code dieser Activity.

Nachfolgende Abbildung zeigt den XML-Code von „activity_main.xml“. Rot markiert sehen Sie den Ausschnitt zur Anzeige des Texts „Hello World!“:

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="ch.nyp.helloworld.MainActivity"
    >
    Ausgabe von "Hello World!" auf dem
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
    />
</android.support.constraint.ConstraintLayout>

```

Abbildung 20 activity_main.xml der HelloWorld-App

drawable

Im Verzeichnis `drawable` werden alle Drawables abgespeichert. Dies sind einerseits alle Bilder (PNG, JPG, GIF), andererseits können dies auch XML-Dateien sein, welche das Aussehen eines bestimmten GUI-Elements generell beeinflussen (kann man sich ähnlich wie ein Stylesheet im Web vorstellen). Möchte ich beispielsweise in meiner App mehrfach einen grünen Button mit runden Ecken haben, so kann im Drawable-Verzeichnis ein XML erstellt werden, welches das Look & Feel dieses Buttons bestimmt. Dieses wird dann im Layout-File bei jedem Button, welcher dieses Look & Feel annehmen soll, angegeben/verlinkt. Weitere Infos dazu folgen im Kapitel 6.3.3

mipmap

Im `mipmap`-Verzeichnis wird das App-Icon, d.h. das Icon welches für die App auf dem Startbildschirm/Home-Bildschirm des Smartphones angezeigt wird.

Generiert werden folgende 4 Ausprägungen von Dateien:

Datei	Beschreibung
<code>ic_launcher.png</code>	<p>Klassisches App-Icon, welches im Home-Bildschirm angezeigt wird.</p> <p>In der HelloWorld-App sehen Sie, dass 5 verschiedene „<code>ic_launcher.png</code>“ vorhanden sind. Dies ist 5 Mal dasselbe Icon, jedoch in einer unterschiedlichen Auflösung. Je nach Bildschirmgrösse des Geräts nimmt Android dann automatisch die richtige Auflösung. Bei Tablets erscheinen Icons beispielsweise grösser als bei Smartphones.</p> <p>Das App-Icon sollte immer den Dateinamen „<code>ic_launcher.png</code>“ tragen.</p> <p>Falls Sie das Icon ersetzen möchten, ersetzen Sie jede Bilddatei mit einer neuen Bilddatei in derselben Grösse.</p>
<code>ic_launcher_round.png</code>	<p>Seit Android 8.0 gibt es sogenannte Adaptive Icons. Dies sind keine statischen Icons mehr, sondern können animiert sein. Auch können diese eine andere Form (z.B. rund) aufweisen. Mehr dazu, siehe https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive.html</p>
<code>ic_launcher.xml</code> <code>ic_launcher_round.xml</code>	<p>Diese beiden Dateien werden für die Anzeige der Adaptive Icons benötigt und werden nur bei Geräten mit Android 8.0 und höher benötigt.</p>

values

Das Value-Verzeichnis beinhaltet folgende Dateien:

Datei	Beschreibung
<code>strings.xml</code>	Texte für die App. Ist das Sprachfile in der Standardsprache der App. Weitere Informationen zu Texten und Mehrsprachigkeit finden Sie im Kapitel 6.2.6.
<code>colors.xml</code>	Definition von Farben, welche dann in den Layouts und Drawables verwendet werden können.
<code>styles.xml</code>	Definiert den Style, d.h. das grundsätzliche Look & Feel der App. In diesem File kann beispielsweise die primäre Farbe, die Hintergrundfarbe von Ansichten, die Farbe der Action Bar, etc. definiert werden.

Tabelle 3 Dateien des Value-Verzeichnis

5.2.3 Manifest-Datei

Die Manifest-Datei ist die zentrale Beschreibungsdatei einer App. Sie befindet sich im Verzeichnis „manifests“ und trägt immer den Dateinamen „AndroidManifest.xml“. In dieser Datei werden Activities, Services, Broadcast-Receiver und Content Provider beschrieben. Jede Activity der App muss in dieser Datei eingetragen werden, sonst kann sie von der App nicht angezeigt werden.

In der Manifest-Datei werden zudem allgemeine App-Einstellungen vorgenommen, d.h. beispielsweise das App-Icon, das Theme (Style) oder die Bezeichnung der App angegeben (wie sie auf dem Smartphone angezeigt wird).

Auch die Berechtigungen, welche eine App benötigt, müssen in dieser Datei angegeben werden. Das kann beispielsweise die Berechtigung zum Schreiben auf das Dateisystem, für den Zugriff auf das GPS oder für den Zugriff auf die Kamera sein.

Nachfolgende Abbildung zeigt die Manifest-Datei der HelloWorld-App:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="ch.nyp.helloworld"
    xmlns:android="http://schemas.android.com/apk/res/android"
    >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        >
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

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

</manifest>
```

Abbildung 21 Manifest-Datei der HelloWorld-App

5.2.4 Gradle Scripts

Hier befinden sich alle Gradle Scripts. In Kapitel 4.5 haben Sie bereits ausführliche Informationen zu Gradle erhalten. Wirklich relevant für Entwickler ist meistens nur die Datei „build.gradle (Module: app)“.

5.3 App im Emulator ausführen

Damit Sie die HelloWorld-App im Emulator testen können, klicken Sie in der Symbolleiste von Android Studio den Play-Button.

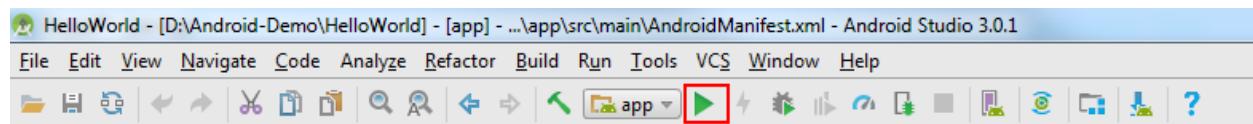


Abbildung 22 HelloWorld-App im Emulator ausführen

Nun erscheint ein Dialog wo Sie unter „**Available Emulators**“ ihre erstellten virtuellen Geräte sehen. Falls Sie noch kein virtuelles Gerät haben können Sie dieses entweder hier über den Button „**Create New Emulator**“ oder über den **AVD-Manager** erstellen (siehe Kapitel 4.4).

Wählen Sie nun das gewünschte virtuelle Gerät aus und klicken Sie den Button „**OK**“ Der Emulator startet nun die App. Dies kann ein paar Minuten dauern.

5.4 App auf Smartphone übertragen

Um die App auf das Smartphone zu übertragen, muss das Smartphone per USB mit dem Computer verbunden und der Smartphone-Treiber auf dem Computer installiert sein.

Zudem muss das USB-Debugging auf dem Smartphone aktiviert sein. Dazu gehen Sie wie folgt vor:

1. Suchen Sie in den Einstellungen ihres Smartphones die Option „Entwickler-Optionen“. Sollten Sie die Option nicht finden, führen Sie Schritt 2 aus. Falls die Entwickler-Optionen angezeigt werden, fahren Sie bei Schritt 3 weiter.
2. Suchen Sie die Option „Geräteinformationen“. Dort finden Sie die Option „Buildnummer“. Tippen Sie 7 Mal auf die Buildnummer. Damit werden die Entwickleroptionen freigeschaltet.
3. Tippen Sie auf die Option „Entwickler-Optionen“. Aktivieren Sie in den Entwickler-Ansicht folgende Optionen:
 - a. Wach bleiben
 - b. Bluetooth HCI Snoop Log
 - c. USB Debugging.

Sobald USB-Debugging aktiviert, das Smartphone verbunden und der Treiber installiert ist, können Sie, wie beim Emulator die App über den Play-Button starten. Im erscheinenden Dialog können Sie nun ihr Smartphone auswählen und die App mit dem Button „OK“ auf dem Smartphone starten.

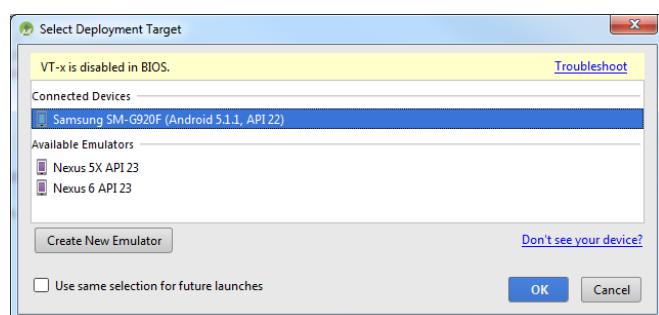


Abbildung 23 HelloWorld-App auf Smartphone übertragen

5.5 Programmcode debuggen

Wie in Eclipse oder in anderen Entwicklungsumgebungen ist es auch in Android-Studio möglich, Programmcode zu debuggen. Alle Java-Klassen können gedebuggt werden. XML-Files lassen sich nicht debuggen.

Zuerst muss, gleich wie in Eclipse, mit einem Mausklick am rechten Rand in der gewünschten Zeile ein Breakpoint gesetzt werden.



```

package ch.nyp.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

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

```

Abbildung 24 Breakpoint in der Klasse „MainActivity“

Die App kann nun über das „**Debug app**“-Icon in der Symbolleiste auf dem Smartphone oder im Emulator gedebuggt werden.

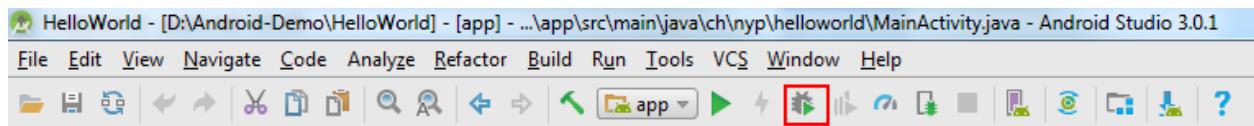


Abbildung 25 HelloWorld-App debuggen

Die App wird nun auf dem Smartphone oder im Emulator (je nach Auswahl) im Debug-Modus gestartet und hält beim gesetzten Breakpoint an.

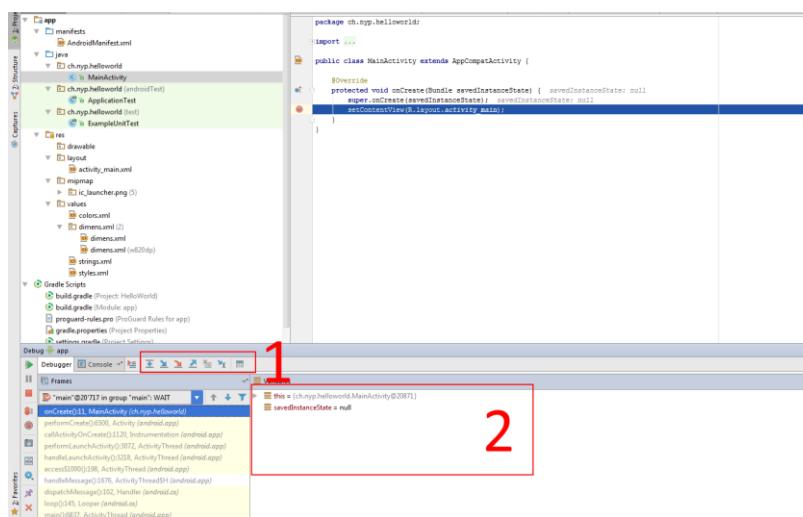


Abbildung 26 HelloWorld-App debuggen 2

- 1) Schritt für Schritt (je nach Wunsch) durch den Code debuggen
- 2) Anzeige der aktuellen Werte der verschiedenen Variablen

6 Benutzeroberfläche gestalten und realisieren

Eine Benutzeroberfläche mit der Entwicklungsumgebung umsetzen ist die eine Sache, dass diese Benutzeroberfläche jedoch nutzerorientiert und intuitiv daher kommt, eine andere. Als Designer und Entwickler von Benutzeroberflächen gilt es einiges zu beachten. So sollten einerseits bei jeder Gestaltung eines Software-GUIs, egal ob für PC oder mobilen Geräte, gewisse ergonomische Grundsätze eingehalten werden. Zum anderen muss sich jeder Designer/Entwickler bewusst sein, dass die Darstellungsmöglichkeiten auf mobilen Geräten limitiert sind und verschiedene mobile Gerättypen mit unterschiedlichen Bildschirmgrößen/Auflösungen existieren.

In diesem Kapitel werden Sie einerseits in die allgemeingültigen theoretischen Grundlagen zur Gestaltung von Benutzeroberflächen eingeführt, andererseits wird ihnen im Detail erklärt, wie Benutzeroberflächen in Android-Studio erstellt werden, welche GUI-Elemente wo gespeichert werden und wie die eigene Android-App zur Anzeige auf Geräte mit verschiedenen Bildschirmgrößen und Ausrichtungen optimiert werden kann.

6.1 Design – Grundlagen

6.1.1 Ergonomische Standards

Die Norm EN ISO 9241 ist ein internationaler Standard, welcher Richtlinien zur Interaktion zwischen Mensch und Computer beschreibt. Diese Norm beinhaltet verschiedene Teile. Interessant für die Gestaltung von Benutzeroberflächen ist der Teil 110 (DIN EN ISO 9241-110) dieser Norm. Diese sagt folgendes:

- **Aufgabenangemessenheit:** Unnötige Interaktionen sollen minimiert werden. Das Fenster und die in diesem Fenster angezeigten Elemente und Texte sollen zur Erledigung der Aufgabe beitragen.
- **Selbstbeschreibungsfähigkeit:** Texte, Meldungen, etc. sollen auf Anhieb verständlich und selbstbeschreibend sein. Der Benutzer soll mit Rückmeldungen über Systemaktionen informiert werden.
- **Lernförderlichkeit:** Die Bedienschritte, Tastenkürzen, der GUI-Aufbau oder Menüeinträge sollen einem einheitlichen, leicht zu verstehenden Prinzip folgen mit dem Ziel: minimale Erlernzeit.
- **Steuerbarkeit:** Schaltflächen, Icons und Menüeinträge sollten den Benutzern mit einfachen und flexiblen Dialogwegen zum Ziel der Aufgabe führen. Bedienungsschritte sollten aufhebbar oder rückgängig gemacht werden können.
- **Erwartungskonformität:** Die Abläufe, Symbole und die Anordnung von GUI-Elementen sollen innerhalb der Anwendung konsistent sein.
- **Individualisierbarkeit:** Fenstereinstellungen, Sortierungen, Symbolleisten, Menüs, Tastenkürzen, Funktionstasten, etc. sollen individuell an die Bedürfnisse und Kenntnisse des Benutzers angepasst werden.
- **Fehlertoleranz:** Das System sollte helfen, Fehler durch Personen möglichst zu vermeiden und Korrekturmöglichkeiten anbieten.

Nicht alle diese Normen können 1:1 auf die Entwicklung von mobilen Apps angewandt werden. Grundsätzlich können die obengenannten Punkte in einer mobilen App wie folgt erreicht werden:

Lernförderlichkeit = Übersichtlichkeit

Die Elemente der Benutzeroberfläche werden nicht willkürlich angeordnet, sondern werden in logischer Abfolge platziert. Benutzer sollen die Benutzeroberfläche von oben nach unten bearbeiten können. Die GUI-Elemente sollen untereinander ausgerichtet sein.

Selbstbeschreibungsfähigkeit = Selbsterklärende Benutzung

Der Benutzer weiss sobald er die Benutzeroberfläche anschaut, was zu machen ist. Dies ist gerade bei mobilen Anwendungen sehr wichtig, weil zu wenig Platz vorhanden ist, um irgendwelche Hilfetexte oder lange Tooltips anzuzeigen. Man spricht hierbei von „Intuitiver Bedienung“.

Auch soll der Benutzer mit Rückmeldungen über erwartete Systemaktionen informiert werden. Klickt er z.B. auf den Button „Beobachtung senden“, was im Hintergrund dazu führt dass eine Beobachtung an den Server gesendet wird, so soll dem Benutzer angezeigt werden, dass die Beobachtung gesendet wird. Sobald die Beobachtung erfolgreich gesendet wurde, soll dies wiederum mit einer Meldung bestätigt werden.

Steuerbarkeit = Bequeme Bedienung

App für mobile Geräte werden mit den Fingern bedient. Daher muss darauf geachtet werden, dass die GUI-Elemente genügend gross sind um mit den Fingern bedient zu werden. Solange nur GUI-Komponenten des Frameworks (bei uns die GUI-Komponenten von Android-Studio) verwendet werden, ist dieser Punkt automatisch erfüllt. Ist jedoch die Entwicklung von eigenen GUI-Komponenten nötig, weil die Standard GUI-Komponenten nicht ausreichen, ist ein besonderes Augenmerk auf dieses Kriterium zu richten.

Aufgabenangemessenheit = Klare Anordnung der GUI-Elemente

Eine Bildschirmseite soll niemals überfrachtet werden. Auf jeder Seite sollen nur Elemente und Texte platziert werden, welche zur Steuerung benötigt werden. Eine Bildschirmseite mit zu vielen GUI-Elementen wird schnell unübersichtlich und kann den Benutzer überfordern. Ist eine Funktion zu komplex für eine Bildschirmseite, so soll ein Schritte-Modell gewählt werden und die Funktionalität damit auf mehrere Bildschirmseiten aufgeteilt werden.

Erwartungskonformität = Einheitlichkeit

Die verschiedenen Ansichten einer App sollen konsistent, d.h. einheitlich sein. Das betrifft den Aufbau, die Farben, die Schriften, die Dimensionen, etc. Bei Android ist es daher wichtig, Dimensionen, Texte und auch Farben in die dafür vorgesehenen Files auszulagern und dann in den Layout-Dateien nur noch einzubinden.

Individualisierbarkeit = Anpassbarkeit = Berücksichtigung der Bildschirmgrößen

Auf dem Markt existiert eine Vielzahl von Android-Geräten mit unterschiedlichen Bildschirmgrösse. Dies ist bei der Erstellung eines App-GUIs zu berücksichtigen. Die eigene App soll ja nicht nur auf einem bestimmten Gerät, sondern auf vielen verschiedenen Geräten optimal angezeigt werden.

6.1.2 Android Design Prinzipien von Google

Das Android User Experience Team hat selber 3 Design-Prinzipien für die Gestaltung von mobilen Anwendungen entwickelt und herausgegeben. Nachfolgend die wichtigsten Punkte der 3 Design-Prinzipien gemäss Beschreibung von Google.

Enchant Me – Verzaubere mich

- Schöne Oberfläche
- sorgfältig platzierte Animationen, gut getimte Soundeffekte
- Berührbare und verschiebbare Objekte
- Benutzer können eine persönliche Note zum GUI hinzufügen (z.B. Profilfotos), App ist selbstlernend, z.B. Favoriten bei Suche.

Simplify My Life – Vereinfache mein Leben

- Kurze, einfache Sätze mit wenigen Worten verwenden
- Bilder sagen mehr als 1000 Worte
- Entscheidungen abnehmen, der Benutzer soll jedoch das letzte Wort haben,
- Nur Anzeigen was wirklich gebraucht wird
- Der Benutzer soll immer wissen, wo er sich befindet. Kleine Animationen bei Bildschirmübergängen. Mit verschiedenen Farben arbeiten.
- Funktional gleiches auch gleich darstellen, funktional anderen, anders darstellen
- Person nur unterbrechen (z.B. Push-Nachricht), wenn es wirklich nötig ist.

Make Me Amazing – Verblüffe mich

- Bei Fehlermeldungen klare Anweisungen geben. Der Benutzer darf nicht das Gefühl haben, er sei schuld am aufgetretenen Fehler.
- Komplexe Aufgabe in kleinere, leichtere Schritte aufteilen. Feedback geben auf Aktionen.
- Die wichtigsten Funktionen sollen schnell zu finden sein

Link Android Design Principles: <http://developer.android.com/design/get-started/principles.html>

6.1.3 Android Material Design von Google

Google hat mit Android 5.0 das Material Design eingeführt. Beim Material Design handelt es sich um einen Leitfaden für die Gestaltung von visuellen Elementen, Bewegungen und Interaktionen in mobilen Anwendungen für verschiedenen Plattformen und Geräte.

Das Material Design von Google basiert auf kartenähnliche Flächen und auf dem Gestaltungsstil „Flat Design“. Dieser Gestaltungsstil ist für seinen Minimalismus bekannt. Die Android-GUI-Komponenten sind eher minimalistisch gehalten, setzen dafür aber auf Animationen und Schatten. Damit wird ein leichter Tiefeneffekt erzeugt und der Nutzer erkennt sofort, auf welche Elemente er drücken kann und welche Bereiche wichtiger sind als andere.

Seit Android 5.0 bietet Android die GUI-Komponenten, Widgets und auch benutzerdefinierte Schatten und Animationen im Android Material Design an.

Als Entwickler ist es wichtig, das Android Material Design zu kennen und falls die Ressourcen vorhanden sind, die eigene App nach Material Design umzusetzen. Dies betrifft vor allem die Realisierung von eigenen GUI-Komponenten.

Im Rahmen dieses Moduls müssen Sie Ihre App jedoch nicht nach Material Design realisieren, falls Sie jedoch möchten, dürfen Sie das gerne tun.

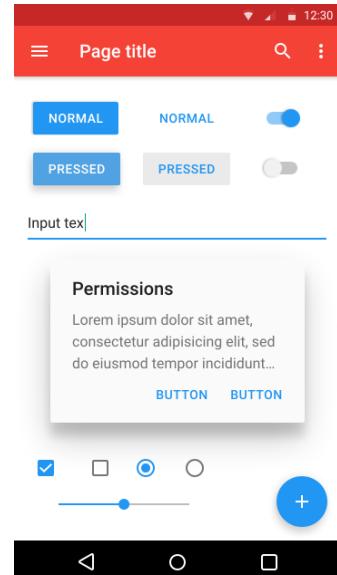


Abbildung 27 Beispiele von GUI-Komponenten mit Material Design. [9]

Link Material Design Guide: <http://developer.android.com/design/material/index.html>

6.2 Benutzeroberfläche für Android realisieren

Nachdem Sie nun mit den Richtlinien für die Gestaltung von intuitiven, benutzerfreundlichen Benutzeroberflächen vertraut sind, werden Ihnen nun die Grundlagen zur Realisierung von GUIs in der eigenen Android-App vermittelt. Im Kapitel 5.2 haben Sie anhand der HelloWorld-App bereits einige Elemente der GUI-Realisierung kennengelernt. Nun werden weitere folgen.

6.2.1 Activity

Was ist eine Activity?

Activities sind die zentralen Elemente jeder Android-App. Eine Activity stellt vereinfacht gesehen eine Bildschirmseite dar. Jede Activity besteht aus einer XML-Datei und einer Java-Datei. In der XML-Datei wird mit Layouts (siehe Kapitel 6.2.2) und Widgets (siehe Kapitel 6.2.5) die Anzeige des GUIs definiert, d.h. es wird das GUI mit GUI-Komponenten wie Textfelder, Buttons, Texte, Bilder, etc. kreiert. Das GUI kann entweder über einen GUI-Designer oder direkt per XML erstellt werden.

Die Java-Datei der Activity managt einerseits den Lebenszyklus der Ansicht, d.h. startet und beendet die Ansicht, lädt das XML-File zur Anzeige des GUIs, nimmt Interaktionen der User entgegen und ruft andere Java-Klassen zur Abarbeitung des Funktionalität (Businesslogik) auf. Jede Activity-Klasse erbt von der Klasse „Activity“ des Android-Frameworks. Weitere Informationen zur Java-Klassen und zum Lebenszyklus von Activities erhalten Sie im Kapitel 0. Für den Moment werden wir uns auf den XML-Teil der Activity konzentrieren.

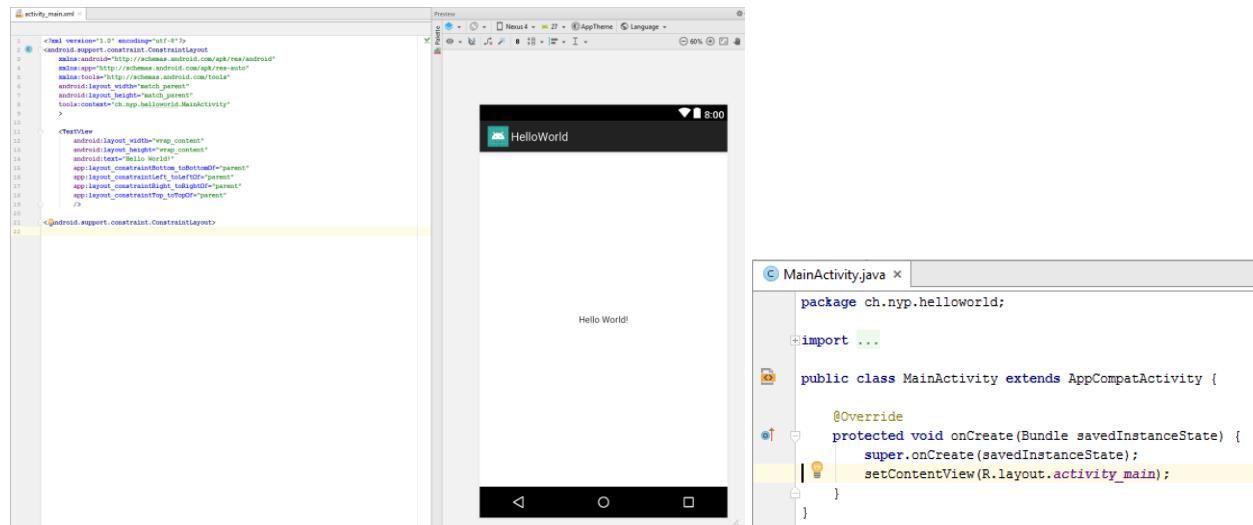


Abbildung 28 Activity mit XML-Datei im Text- und Designmodus, sowie Java-Code der Activity

Jede App hat eine Haupt-Activity, die sogenannte MainActivity. Diese wird beim App-Start aufgerufen. Die MainActivity ruft dann, sobald eine andere Ansicht angezeigt werden soll, eine andere Activity auf, usw. Der Wechsel einer Activity ist für den Benutzer durch eine kurze Animation sichtbar. Wenn eine Activity eine neue Activity zur Anzeige aufruft, so wird die alte Activity vom System angehalten und auf einen Stapel, den sogenannten „Back Stack“ gelegt. Drückt der Benutzer nun in der neuen Activity den Zurück-Button in der Action Bar, oder die neue Activity wird aus einem anderen Grund beendet, so wird die oberste Activity im Back Stack gestartet, d.h. wieder die alte Activity angezeigt. Jede neue Activity, welche gestartet wird überlagert damit die alte Activity. Diese ist aber im „Back Stack“ weiterhin vorhanden.

Nachfolgende Abbildung zeigt das Starten und Beenden von Activities am Beispiel der XelHa-App.

Modul 335: Mobile-Applikationen realisieren

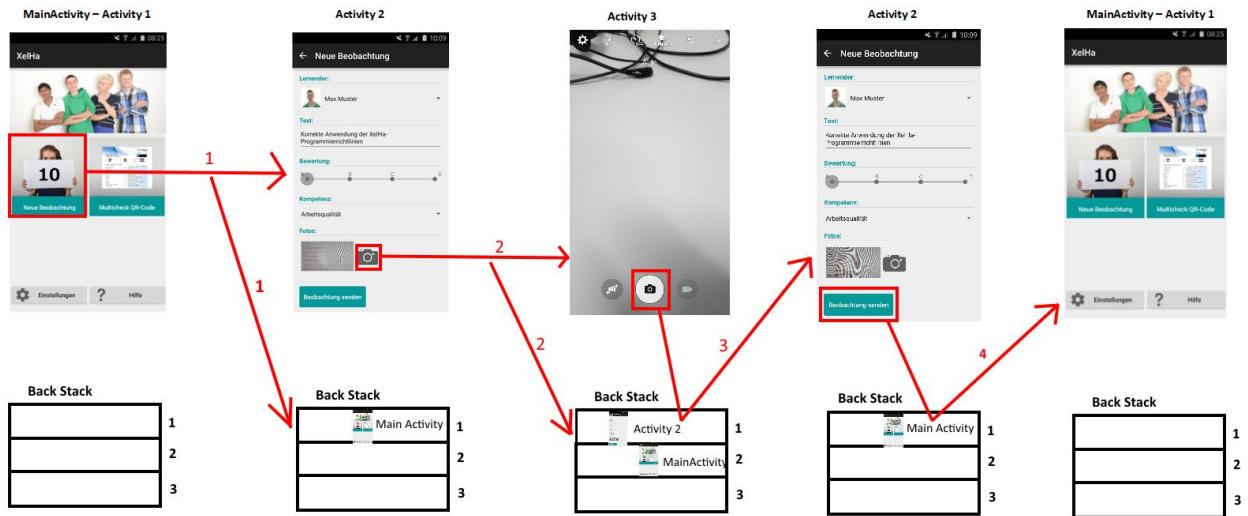


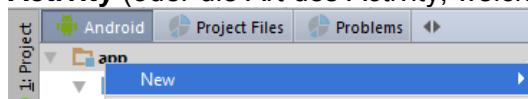
Abbildung 29 Starten und Beenden von Activities am Beispiel der XelHa-App

- Der Benutzer klickt auf den Button „Neue Beobachtung“. Das System startet nun die Activity 2 und legt die Activity 1 auf den Back Stack.
- Der Benutzer klickt auf den Foto-Button um ein Foto zu machen. Das System startet die Activity 3 zum Schiessen eines Fotos. Die Activity 2 kommt zuoberst auf den Back Stack. Die Main Activity wird eine Position nach unten geschoben im Back Stack.
- Der Benutzer schiesst ein Foto. Das System schiesst das Foto und beendet die Activity 3. Damit nun wieder eine Activity angezeigt wird holt das System automatisch die oberste Activity vom Back Stack, in unserem Fall die Activity 2, und zeigt diese an.
- Der Benutzer sendet die Beobachtung mit dem Button „Beobachtung senden“ ab. Die App sendet die Beobachtung an einen Webservice und schliesst die Activity 2. Das System holt wieder die oberste Activity vom Back Stack und zeigt somit wieder die Main Activity an.

Activity erstellen

Um Ihrem Projekt eine neue Activity hinzuzufügen gehen Sie wie folgt vor:

1. Rechtsklick auf „app“ im Android Projekt Explorer und dann „New → Activity → Empty Activity“ (oder die Art des Activity, welche gewünscht wird)



2. Nun können Sie den Activity Name, den Layout Name und der Package Name angeben. Wählen Sie immer einen aussagekräftigen Activity Name und nicht Activity1 oder so. Der Activity Name entspricht dem Name der Activity-Java-Klasse. Dieser soll immer mit „Activity“ enden. Der Layout Name ist der Dateiname des XML-Files vom Activity. Dieser soll mit „activity“ beginnen und die Wörter mit Underline (_) voneinander getrennt werden. Für das Activity zum Erfassen einer neuen Beobachtung beispielsweise empfehlen sich folgende Angaben:

Activity Name: NewBeobachtungActivity

Layout Name: activity_new_beobachtung

Launcher Activity: Nein. Muss Aktiviert sein, wenn es sich um das Activity, welches beim Start der App aufgerufen werden soll, handelt (MainActivity).

Package Name: Name des Package, in welchem das Java-File der Activity abgelegt werden soll.

3. Klicken Sie den Button „Finish“. Android Studio erstellt nun die Activity, d.h. die Java-Klasse und auch die XML-Klasse. Die Layout Datei (XML) beinhaltet nur die Angabe des Layouts (mehr siehe Kapitel 6.2.2), ist sonst jedoch leer. Android-Studio generiert den Code je nach gewähltem Activity-Typ beim Schritt 1.

Activity in der Manifest-Datei registrieren

Wie Sie bereits in Kapitel 5.2.3 gelesen haben, besteht jede App aus einer zentralen Beschreibungsdatei, der sogenannten Manifest-Datei. Diese trägt den Dateinamen „AndroidManifest.xml“ und befindet sich im Verzeichnis „manifests“.

Jede Activity einer App muss im Manifest-File registriert werden, sonst kann das System, respektive die App, die Activity nicht anzeigen. Die Activity, welche beim App-Start aufgerufen werden soll (MainActivity) muss zudem im Manifest-File als Launcher- und Main-Activity markiert werden.

Wenn Sie eine Activity, wie unter „Activity erstellen“ von Android Studio generieren lassen, erstellt Android Studio automatisch den Activity-Eintrag im Manifest-File.

Die Activities werden im Manifest-File mit dem Tag `<activity ...>` innerhalb des Application-Blocks definiert. Pflicht ist die Angabe des Namens der Activity (z.B. `android:name=".MainActivity"`). Dies ist der Klassenname der Activity-Java-Klasse.

Nachfolgender Code-Ausschnitt registriert die Main- und eine weitere Activity und muss innerhalb von `<application></application>` platziert werden:

```

<activity android:name=".MainActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
<activity android:name=".NewBeobachtungActivity"></activity>

```

6.2.2 Fragments

Was ist ein Fragment?

Wie Sie im vorhergehenden Kapitel gesehen haben, entspricht eine Activity einer App immer einer Ansicht, welche den ganzen Bildschirm ausfüllt. Mehrere Activities auf einer Bildschirmseite darzustellen ist nicht möglich.

Zudem funktionieren Activities mit dem erläuterten Back Stack. Jede neue Activity die gestartet wird, legt sich über die alte Activity. Die alte Activity bleibt aber im Back Stack vorhanden und wird, sobald die neue Activity geschlossen wird, wieder angezeigt.

Seit Android 3.0 existieren Fragments. Dies sind Bausteine innerhalb von Activities mit einer eigenen Benutzeroberfläche und einem eigenen Lebenszyklus. Die Idee von Fragments ist hauptsächlich, die grosse Bildschirmgrösse von Tablets optimal auszunutzen, indem bei Tablets 2 Ansichten der App auf einer Bildschirmseite dargestellt werden.

Ein Beispiel ist eine App mit einer Listenansicht und wenn auf einen Listeneintrag geklickt wird, erscheint die Detailseite des geklickten Eintrags. Auf dem Smartphone wird zuerst die Liste Bildschirmfüllend angezeigt und nach dem Klick die Detailansicht. Auf dem Tablet kann die Liste und die Detailansicht gleichzeitig angezeigt werden (immer die Detailansicht des selektierten Eintrags).

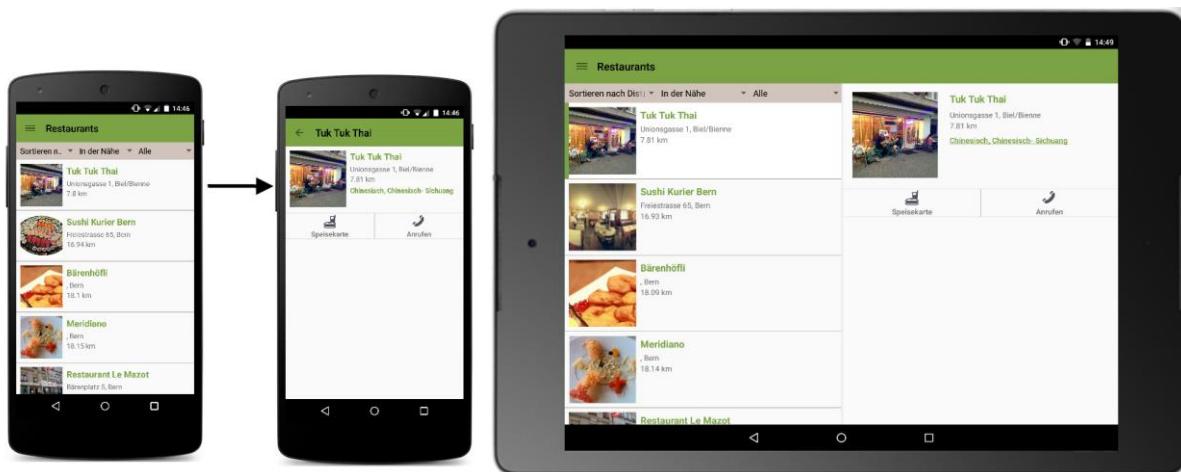


Abbildung 30 List-Detail-Ansicht von Smartphone und Tablet mit Hilfe von Fragments

Möchte eine solche Ansicht realisiert werden, so müssen Fragments verwendet werden. Die Liste und auch die Detail-Ansicht werden in diesem Fall komplett als Fragments und nicht als Activity realisiert. Activities braucht es aber weiterhin. Die Activity ist jedoch nur noch den Container zur Ausführung des Fragments und beinhaltet selbst keine GUI-Elemente mehr. Bei der App aus der obenstehenden Abbildung beispielsweise wird beim Smartphone jeweils ein Fragment in die leere Activity geladen. Beim Tablets werden 2 Fragments in eine Activity geladen und beide nutzen jeweils die Hälfte des Platzes (der Platz ist konfigurierbar im Programmcode).

Modul 335: Mobile-Applikationen realisieren

Ein weiterer Unterschied zwischen Fragments und Activities ist der Back Stack. Möchte ich in meiner App die Ansicht wechseln, ohne dass die neue Ansicht über die alte „drübergelegt“ und die alten zum Back Stack hinzugefügt wird, wähle ich Fragments für die Ansichten.

Dies ist beispielsweise bei einer App mit einem sogenannten „Navigation Drawer“ der Fall. Ein Navigation Drawer ist etwas wie ein Menü. In diesem Menü kann die Ansicht gewechselt werden. Wenn das Menü gewechselt wird soll die alte Ansicht durch die neue ersetzt werden, ohne dass die neue Ansicht nur über die alte gelegt wird, die alte im Back Stack aber weiterhin existiert. Das geht nur mit Fragments.

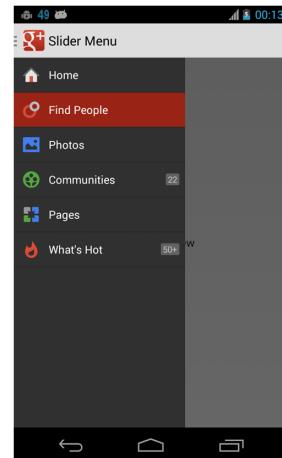


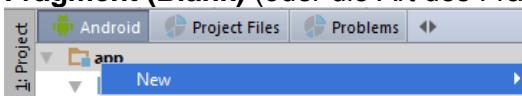
Abbildung 31 Navigation Drawer [10]

Fragment erstellen

Ein Fragment besteht wie eine Activity aus einer Layout-Datei (XML) und einer Java-Klasse. Für die Gestaltung des GUIs stehen dieselben GUI-Komponenten (Layout, Widgets, etc.) und Gestaltungsmöglichkeiten wie beim Activity zur Verfügung.

Um ihrem Projekt ein neues Fragment hinzuzufügen gehen Sie wie folgt vor:

1. Rechtsklick auf „app“ im Android Projekt Explorer und dann „New → Fragment → Fragment (Blank) (oder die Art des Fragments, welche gewünscht wird)“



2. Nun können Sie den Fragment Name und den Layout Name angeben. Wählen Sie immer einen aussagekräftigen Fragment Name und nicht Fragment1 oder so. Der Fragment Name entspricht dem Name der Fragment-Java-Klasse. Dieser soll immer mit „**Fragment**“ **enden**. Der Layout Name ist der Dateiname des XML-Files vom Fragment. Dieser soll mit „**fragment**“ **beginnen** und die Wörter mit Underline (_) voneinander getrennt werden. Für das Fragment zum Erfassen einer neuen Beobachtung beispielsweise würden sich folgende Angaben empfehlen:

Fragment Name: NewBeobachtungFragment

Layout Name: fragment_new_beobachtung

Die Checkboxen bei „**Include fragment factory methods?**“ und „**Include interface callbacks**“ sollten **nicht aktiviert** sein, ausser wenn gewünscht wird, dass gewisser Code für den Aufruf und das Handling des Fragments bereits generiert wird. Meistens empfiehlt sich aber den Code von Grund auf selbst zu implementieren.

3. Klicken Sie den Button „**Finish**“. Android Studio erstellt nun das Fragment, d.h. die Java-Klasse und auch die XML-Klasse.

Das generierte Layout-File des Fragments befindet sich im Verzeichnis „res/layout“, die Java-Klasse im Root-Package des Java-Verzeichnisses. Fragments müssen nicht in der Manifest-Datei eingetragen werden. Gestartet werden vom Android-System immer noch die Activities, welche dann in dem Activity-Java-Code die entsprechenden Fragments aufrufen (mehr dazu später in diesem Dokument).

6.2.3 GUI von Activity oder Fragment realisieren

Nun haben Sie viel theoretisches Wissen über Activities und Fragments erlangt und möchten endlich wissen wie diese GUIs erstellt werden können.

Wie erläutert, wird das GUI von Activities oder Fragments entweder per XML oder über den Designer (oder beides) realisiert.

GUI-Designer

Wenn Sie auf das XML-File des Fragments doppelklicken, erscheint der GUI-Designer. Sie können das GUI komplett über den GUI-Designer realisieren.

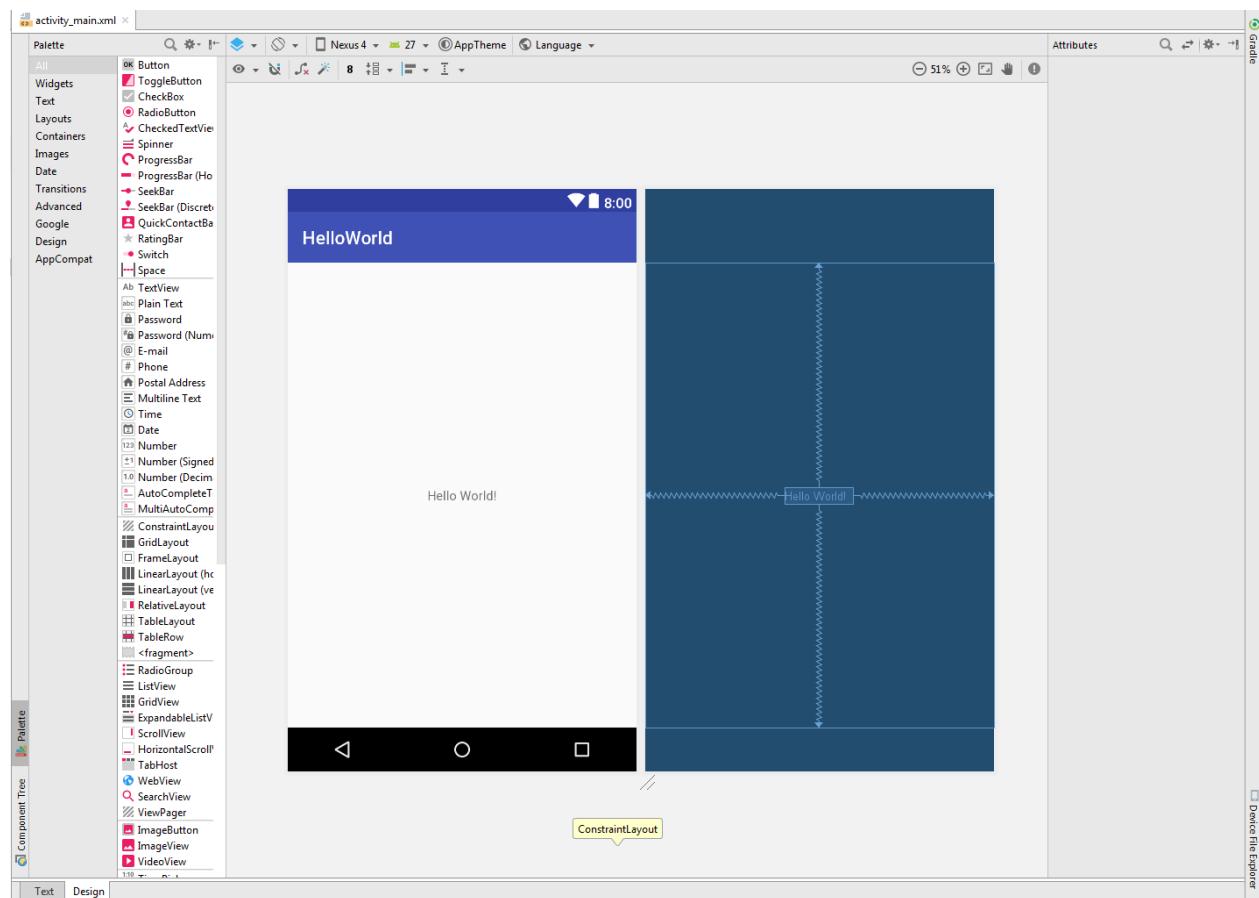


Abbildung 32 GUI-Designer für Realisierung von Activity/Fragment

Der GUI-Designer besteht aus folgenden Elementen:

- **Palette:** Links sehen Sie die Palette mit GUI-Komponenten. Hier können Sie Textfelder, Layouts, Buttons, etc. auf das GUI ziehen.
- **Vorschau:** In der Mitte sehen Sie die Vorschau der Ansicht. Oberhalb der Vorschau können Sie mit Buttons die Orientierung, das Smartphone, die Android-Version, etc. für die Vorschau ändern um bereits einen ersten Einblick zu erhalten, wie das GUI auf einem anderen Smartphone/Tablet oder im Querformat aussehen wird.
- **Attributes:** Hier können die Einstellungen des in der Vorschau selektierten GUI-Elements verändert werden. Z.B. die ID des Elements definieren, den Text bei einer TextView (sowas wie ein Label) angeben, Höhe und Breite des Elements angeben und noch viel mehr.

Modul 335: Mobile-Applikationen realisieren

XML

Über das Tab „Text“ können Sie in die XML-Ansicht wechseln um den XML-Code des GUIs zu sehen oder direkt in XML das GUI zu programmieren.

In der XML-Ansicht ist es auch möglich, ein Preview anzusehen. Klicken Sie dazu am rechten Rand auf „Preview“.

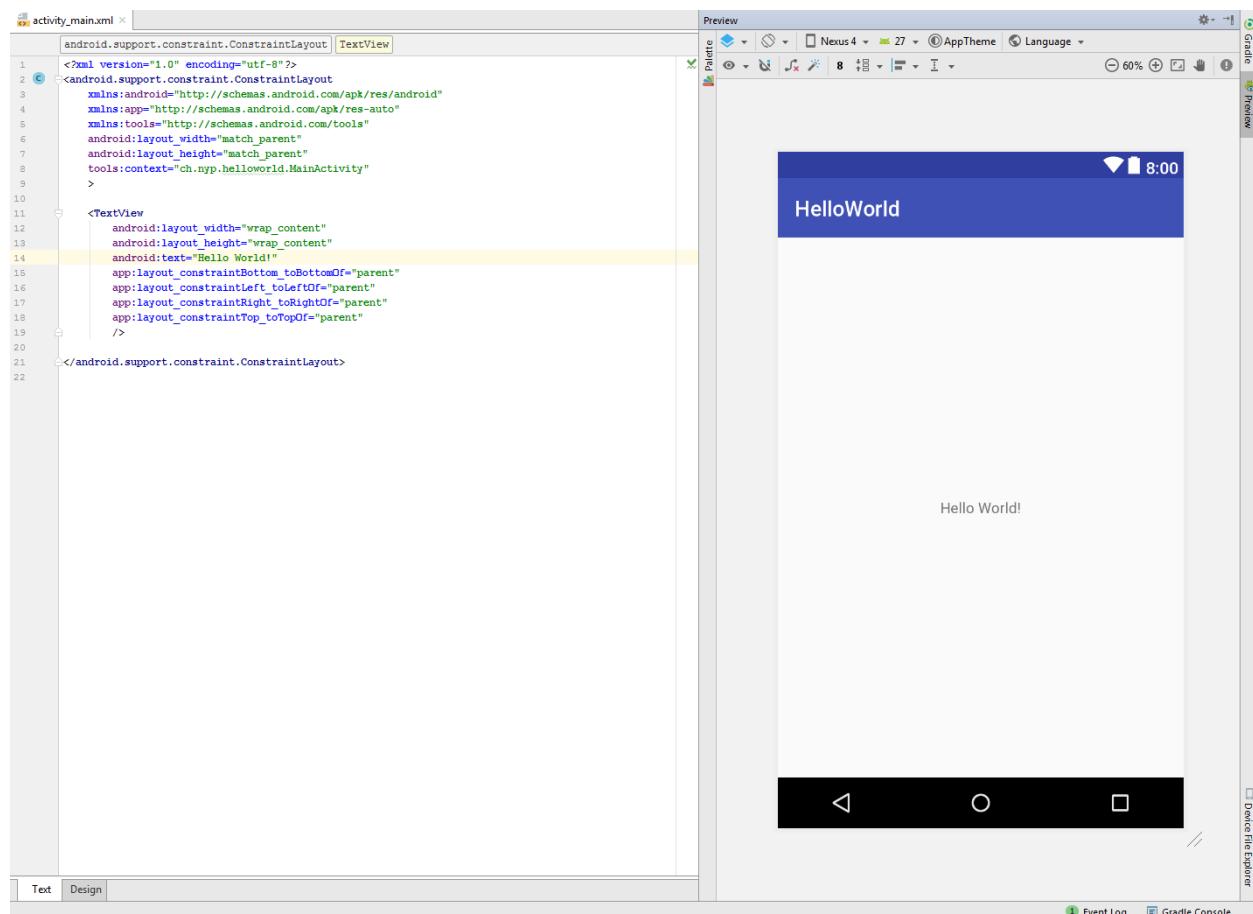


Abbildung 33 XML-Ansicht für Realisierung von Activity/Fragment

6.2.4 Layouts

Die Grundlage jedes GUIs bilden Layouts. Das Root-Element jedes Fragments und jeder Activity muss ein Layout sein. Layouts sind dazu da, GUI-Elemente anzugeordnen. Layouts können ineinander verschachtelt werden.

Die wichtigsten Layouts sind:

LinearLayout

Ordnet die GUI-Elemente entweder horizontal (nebeneinander) oder vertikal (untereinander) an. Die Ausrichtung (horizontal oder vertikal) wird über den XML-Parameter **android:orientation**, oder im GUI-Designer bei den Properties des Layouts, festgelegt.

Die detaillierte Anordnung der Elemente geschieht mit Eigenschaften wie `layout_marginXXX` (z.B. `layout_marginTop`, `layout_marginBottom`, etc.) mit Padding, mit Breite und Höhe der Elemente (`layout_width` / `layout_height`) etc. Elemente werden so in Bezug auf andere Elemente ausgerichtet.

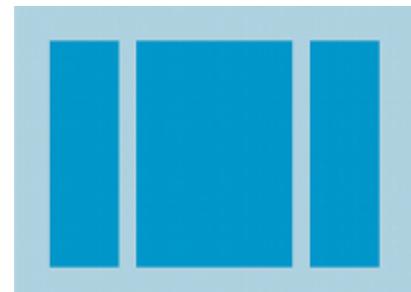


Abbildung 34 horizontales LinearLayout

RelativeLayout

Ordnet die GUI-Elemente relativ zueinander an. Es kann also für ein GUI-Element z.B. definiert werden, dass es Rechts vom GUI-Element X steht und unterhalb von GUI-Element Y.



Abbildung 35 RelativeLayout

ConstraintLayout

Das ConstraintLayout existiert seit Android 2.3, wird aber erst seit Android Studio 3.0 (Oktober 2017) vollkommen in der Entwicklungsumgebung unterstützt. Es ist dem RelativeLayout sehr ähnlich, bietet aber eine weit höhere Flexibilität und ermöglicht auch komplexe GUI's, die vorher nur in Kombination von RelativeLayout und LinearLayout möglich waren. Zudem bietet es eine flache Struktur – es muss nicht verschachtelt werden (nur 1 Hierarchiestufe). Daher ist die Performance deutlich besser als bei den anderen Layouts.

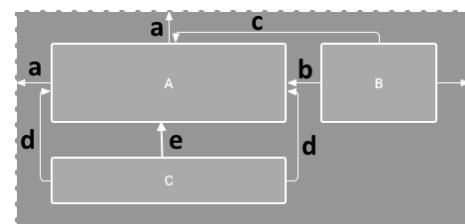


Abbildung 36 ConstraintLayout

Im Constraint Layout werden GUI-Elemente durch Verbindungen zueinander ausgerichtet (siehe Pfeile in Abbildung). In der Abbildung sind die GUI-Elemente wie folgt verbunden:

- GUI-Element A ist das erste GUI-Element. Es ist mit sogenannten Parent-Positionen (Pfeile „a“) mit dem GUI-Layout verbunden.
- GUI-Element B ist mit der Verbindung „b“ rechts von Element B ausgerichtet, d.h. es erscheint immer Rechts vom GUI-Element B. Nur mit dieser Verbindung ist B aber noch nicht zwingend auf denselben Höhe wie A.
- Durch die Verbindung „c“ wird nun festgelegt, dass sich GUI-Element A und B auf denselben Höhe befinden.
- Durch die Verbindungen „d“ ist definiert, dass GUI-Element C und A immer gleich breit angezeigt werden. Nur mit den Verbindungen „d“ ist aber noch nicht gewährleistet, dass Element C unterhalb von A angezeigt wird. Dazu braucht's noch die Verbindung „e“

Weitere Layouts und weiterführende Informationen finden Sie auf folgenden Webseiten:

Linear und RelativeLayout: <http://developer.android.com/guide/topics/ui/declaring-layout.html>

Diverse Layouts: <http://individuapp.com/de/android-kurs/android-layout-klassen>

ConstraintLayout:

<https://developer.android.com/training/constraint-layout/index.html>

<http://www.app-entwicklung.info/2016/12/hallo-constraintlayout/>

Beispielprojekte im GitHub-Repository:

Projekt	Beschreibung
LinearLayoutExample	Aufbau eines GUIs mit dem LinearLayout. Zeigt einen Text, 2 Labels, 2 Eingabefelder und 2 Buttons an.
RelativeLayoutExample	Aufbau eines GUIs mit dem RelativeLayout. Zeigt dasselbe GUI wie LinearLayoutExample.
ConstraintLayoutExample	Aufbau eines GUIs mit dem ConstraintLayout. Zeigt dasselbe GUI wie LinearLayoutExample.

6.2.5 Widgets / Text Fields

Innerhalb eines Layouts werden die GUI-Elemente platziert. Diese GUI-Elemente sind entweder Widgets oder Text Fields. Text Fields sind verschiedene Textfelder (normaler Text, Datum-Feld, Nummer-Feld, E-Mail, etc. Widgets sind beispielsweise Buttons, Checkboxen, Progressbars, ImageView (Anzeige von Bildern).

Am besten platzieren Sie mit dem Designer oder per XML einmal selbst verschiedene Widgets und TextFields in ihrem Layout (Fragment) um zu sehen, welches Widget und TextField wie aussieht.

Weitere Infos zu den wichtigsten Widgets und Text Fields finden Sie auf folgender Webseite:

<http://developer.android.com/guide/topics/ui/controls.html>

6.2.6 Texte

Bei Text Views, bei Buttons und auch andere Widgets beinhalten Texte (Strings), welche auf dem Bildschirm angezeigt/ausgegeben werden.

Es ist möglich, diese Texte direkt ins XML-File oder auch in die Java-Klassen zu schreiben. Dieses Vorgehen ist jedoch nicht zu empfehlen, da Texte dann einerseits nicht mehrfach wiederverwendet werden können, andererseits Texte nicht übersetzt werden können.

Aus diesem Grund gibt es die Value-Datei „**strings.xml**“, die sich im Verzeichnis „**res/values**“ befindet.

Diese Datei soll jeder Text (String) in der Standardsprache des App beinhaltet. D.h. möchte ich meine App nur auf Deutsch verfügbar mache, so trage ich die deutschen Texte in dieser Datei ein. Möchte ich eine mehrsprachige App haben, so kann ich weitere String-Dateien für die entsprechende Sprachen erstellen.

Dazu wird für jede Sprache ein Value-Verzeichnis und in jedem Value-Verzeichnis die Datei strings.xml erstellt. z.B.

- Verzeichnis **values-de** mit strings.xml für **Deutsch**
- Verzeichnis **values-en** mit strings.xml für **Englisch**
- Verzeichnis **values-fr** mit strings.xml für **Französisch**
- Verzeichnis **values-it** mit strings.xml für **Italienisch**

Nachfolgend ein Auszug aus der String-Datei der XelHa-App:

```
<resources>
    <string name="main.button.beobachtung.new">Neue Beobachtung</string>
    <string name="main.button.qrcode.scan">Multicheck QR-Code</string>
</resources>
```

Dies ist ein sogenanntes Key-Value-File. Jeweils eine **<string..> </string>** Block entspricht einem String (Text). Der Name des Strings muss eindeutig sein und wird im XML-Layout-File wo der String angezeigt werden soll, angegeben. Zwischen den String-Tags folgt dann der Text, wie er auf dem Bildschirm angezeigt werden soll. Im string.xml sind dies die Texte in der Standardsprache der App, im strings.xml im Verzeichnis values-de die deutschen Texte, usw. Der „name“ der Strings ist in jedem Sprachfile gleich.

Die Sprache in welcher die App angezeigt wird entspricht der Betriebssystemssprache des Smartphones auf welcher die App ausgeführt wird. Ist kein Sprachfile für die Betriebssystemssprache vorhanden, werden die Texte der Standardsprache (strings.xml im values-Verzeichnis) angezeigt.

Im Layout-XML wird der Text aus dem String-File dann mit **@string/** gefolgt vom **name des Strings aus dem String-File**, eingefügt.

```
<Button
    android:id="@+id/button_main_newbeobachtung"
    android:text="@string/main.button.beobachtung.new"
/>
```

Texte können innerhalb des String-Files formatiert werden (Fett, Kursiv, Unterstrichen, Zeilenumbrüche, etc.). Zudem ist es auch möglich in einem String Platzhalter hinzuzufügen, welche während der Ausführung der App mit dynamischen Angaben (z.B. Zahlen) gefüllt werden.

Weitere Infos dazu finden Sie unter folgendem Link:

<http://developer.android.com/guide/topics/resources/string-resource.html>

6.2.7 Dimensionen

Bei den Dimensionen, d.h. Größenangaben von Layouts, Widgets, etc. ist es ähnlich wie bei den Sprachfiles. Diese können in extra dafür vorgesehene XML-Files ausgelagert werden, den **dimens**-Files. Dimensionen können Schriftgrößen, Abstände, Breite und Höhenangaben, etc. sein. Einfach alles was irgendeine Größenangabe (z.B. in Pixel oder andere Einheit) hat.

Dimensionen sollten in die Dimens-Files ausgelagert werden, damit später für unterschiedliche Bildschirmgrößen und Ausrichtungen verschiedene Dimensionen hinterlegt werden können. Auf einem Tablet möchte ich beispielsweise eine Schrift grösser darstellen als auf einem Smartphone.

Für den Moment reicht es wenn Sie das Standard Dimens-File kennen. Dieses befindet sich unter „**res/values/dimens.xml**“.

Nachfolgende ein Auszug aus einem Dimens-File. Eine Dimension wird mit dem Tag „**<dimen></dimen>**“ definiert, trägt einen Namen, mit welcher Sie in den Layout-Dateien eingebunden wird und die Größenangabe zwischen den Dimen-Tags.

```
<resources>
  <dimen name="height_button_small">16dp</dimen>
  <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

Größenangaben können in Pixels, Inches, Millimeter, Punkte, Density-independent Pixels oder Scale independent Pixels erfolgen. Empfohlen ist die Verwendung von Density-independent Pixels (dp) für Abstände und Größen von Containern und Scale-independent Pixels (sp) für Schriftgrößen.

- **Pixel (px):** Nimmt Bezug zu den Pixel auf dem Bildschirm. Es kann jedoch sein, dass zwei Smartphones, welche in cm den gleich grossen Bildschirm haben (z.B. 8 x 5 cm) eine unterschiedliche Pixelanzahl umfassen, somit die GUI-Elemente dann in unterschiedlicher Größe dargestellt werden.
- **Inches (in):** Größenangabe in Inches. Nimmt Bezug auf die physikalische Bildschirmgröße.
- **Millimeter (mm):** Größenangabe in mm. Nimmt Bezug auf die physikalische Bildschirmgröße.
- **Punkte (pt):** 1pt = 1/72 von einem Inch. Nimmt Bezug auf die physikalische Bildschirmgröße.
- **Density-independent Pixels (dp):** Eine Abstrakte Einheit, welche Bezug nimmt zur physikalischen Dichte des Bildschirms. Die Einheit steht im Verhältnis zu einem 160 dpi (dots per inch) Bildschirm. Auf einem 160dpi Bildschirm ist 1 dp = 1 px, Auf einem 320 dpi Bildschirm entspricht 1dp = 2px, usw. Wichtig ist, dass dp zu pixel nicht direkt proportional umgerechnet wird, sondern nach einem bestimmten Raster (in bestimmten Schritten). Die Schritte entsprechen den für Android-Smartphones möglichen Bildschirmdichten. DP wurde eingeführt, um GUI-Elemente optimal auf die Größe unterschiedlicher Bildschirme zu skalieren. Mehr dazu finden Sie im Kapitel 6.3.1.
- **Scale-independent Pixels (sp):** Wie dp, aber für die optimale Skalierung von Schriftgrößen.

In den XML-Files der Layouts werden Größenangaben aus dem Dimens-File wie folgt angesprochen (Beispiel eines Buttons):

```
<Button
  android:layout_height="@dimen/height_button_small"/>
```

6.2.8 Fragments zu einem Activity hinzufügen

Folgendes Google-Tutorial zeigt gut auf, wie Fragments zu einer Activity hinzugefügt werden können.

<http://developer.android.com/training/basics/fragments/creating.html#AddInLayout>

6.3 Benutzeroberfläche optimieren

6.3.1 Verschiedene Bildschirmgrößen

Einleitung

Für Android existiert eine Vielzahl an Smartphones und Tablets verschiedener Hersteller mit verschiedenen Bildschirmgrößen und Auflösungen. Als App-Hersteller sollte man das Ziel verfolgen, seine App auf möglichst vielen verschiedenen Geräten optimal darstellen zu können.



Abbildung 37 Android-Geräte mit verschiedenen Bildschirmgrößen und Auflösungen [11]

Die Zahl der verschiedenen Bildschirmgrößen und Auflösungen ist unüberschaubar. Daher gibt's bei Android einerseits die im vorhergehenden Kapitel erläuterten Größenangaben „**dp**“ und „**sp**“. Größenangaben von GUI-Elementen, welche in **dp** oder **sp** gemacht werden, werden von Android automatisch für unterschiedliche Bildschirmgrößen skaliert. Daher immer **dp** und **sp** verwenden.

Zudem hat Android die verschiedenen Bildschirme in **6 generalisierte Pixeldichten** aufgeteilt. Diese sogenannte „**Density**“ ist die Anzahl Pixel innerhalb eines physikalischen Bereichs auf dem Bildschirm und wird normalerweise **DPI** (Dots per Inch) genannt. Je grösser der DPI-Wert eines Bildschirms, desto schärfer ist das Bild. Meistens gilt auch: Je neuer das Smartphone, desto höher ist der DPI-Wert des Bildschirms.

Generalisierte Pixeldichten bei Android:

- **mdpi (medium)**: ca. 160 dpi, heute kaum noch Smartphones mit dieser Dichte im Umlauf.
- **hdpi (height)**: ca. 240 dpi
- **xhdpi (extra-high)**: ca. 320 dpi
- **xxhdpi (extra-extra-high)**: ca. 480 dpi
- **xxxhdpi (extra-extra-extra-high)**: ca. 640 dpi

Die genannten Pixeldichten haben einen grossen Einfluss auf die Bilder, die in einer App angezeigt werden. Bilder haben immer eine Grösse in Pixel (Breite x Höhe). Dasselbe Bild wird nun aber auf einem xxxhdpi-Bildschirm viel kleiner dargestellt als auf einem mdpi-Bildschirm, weil beim xxxhdpi-Bildschirm auf einer bestimmten Grösse des Bildschirms (z.B. 1 Quadratmeter) viele mehr Pixel angezeigt werden können. Daher müssen Bilder für die verschiedenen generalisierten Pixeldichten in unterschiedlicher Grösse hinterlegt werden.

Bilder in unterschiedlicher Grösse hinterlegen

Bilder werden im Android-Projekt im Verzeichnis „**res/drawable**“ hinterlegt. App-Icons im Verzeichnis „**res/mipmap**“.

Um Bilder für die verschiedenen generalisierten Pixeldichten zu hinterlegen muss im res-Verzeichnis für jede generalisierte Pixeldichte ein drawable-Verzeichnis erstellt werden, in welchem alle Bilder in der Grösse für die jeweilige Pixeldichte hinterlegt werden.

Das bedeutet, im res-Verzeichnis müssen folgende drawable-Verzeichnisse erstellt werden (siehe Ansicht „Project Files“):

- **drawable-mdpi**
- **drawable-hdpi**
- **drawable-xhdpi**
- **drawable-xxhdpi**
- **drawable-xxxhdpi**

In jedem Verzeichnis muss dann die Bilddatei in der jeweiligen Grösse hinterlegt werden. Der Dateiname der Bilddatei muss in jedem Verzeichnis derselbe sein.

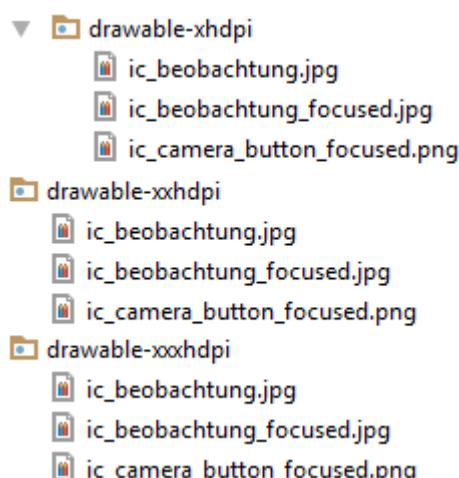


Abbildung 38 Beispiel von Bildern für unterschiedliche generalisierte Pixeldichten.

Das Prinzip bei den App-Icons ist dasselbe. Erstellt werden hier die Verzeichnisse mipmap-hdpi, mipmap-xhdpi, usw.

Für die Umrechnung der Pixel ist folgender Rechner hilfreich:
http://labs.rampinteractive.co.uk/android_dp_px_calculator/

Verschiedene Layouts für unterschiedliche Bildschirmgrößen

Die Anwendung von dp, sp und Bildern in unterschiedlichen Größen reicht meistens aus, damit eine App auf verschiedenen Bildschirmgrößen einigermassen gut dargestellt wird. Sollte eine App jedoch den Platz auf grösseren Bildschirmen optimal ausnutzen und bei grösseren Bildschirmen z.B. mehr Elemente oder Funktionen anzeigen, so gibt es noch die Möglichkeit, Grössenspezifische Layout-Files zu hinterlegen.

Im Normalfall gibt es in einem Android-Projekt ein Verzeichnis „**layout**“, wo die Layout-Files (XML) von Fragments, Activities, etc. abgelegt werden.

Es ist nun jedoch möglich, grössenspezifische layout-Verzeichnisse zu erstellen. Dort kann dann ein nur für diese Grösse gültiges Activity- oder Fragment-XML-File hinterlegt werden.

Weitere Infos dazu finden Sie unter:

http://developer.android.com/guide/practices/screens_support.html#DeclaringTabletLayouts

Für den ÜK reicht es, wenn Sie wissen dass es die Möglichkeit gibt, solche grössenspezifischen Layout-Files zu hinterlegen. Sie müssen dies jedoch in ihrer App nicht umsetzen, das würde den zeitlichen Horizont des ÜKs sprengen.

Weitere Tipps (Best Practises von Google)

Google hat 4 Best Practises rausgegeben, wessen Einhaltung helfen kann, dass die App optimal auf unterschiedlichen Bildschirmgrößen dargestellt wird.

http://developer.android.com/guide/practices/screens_support.html#BestPractices

6.3.2 Quer- und Hochformat

Im Querformat steht, sowohl bei Tablets als auch bei Smartphones, in der Breite viel mehr Platz zur Verfügung als im Hochformat.

Im Zusammenhang mit Fragments im Kapitel 6.2.2 wurde bereits erwähnt, dass es bei Tablets im Querformat üblich ist, mehrere Ansichten (z.B. Liste und Detail) auf einer Bildschirmseite anzuzeigen. Dieser Ansatz kann auch bei Smartphones im Querformat gewählt werden.

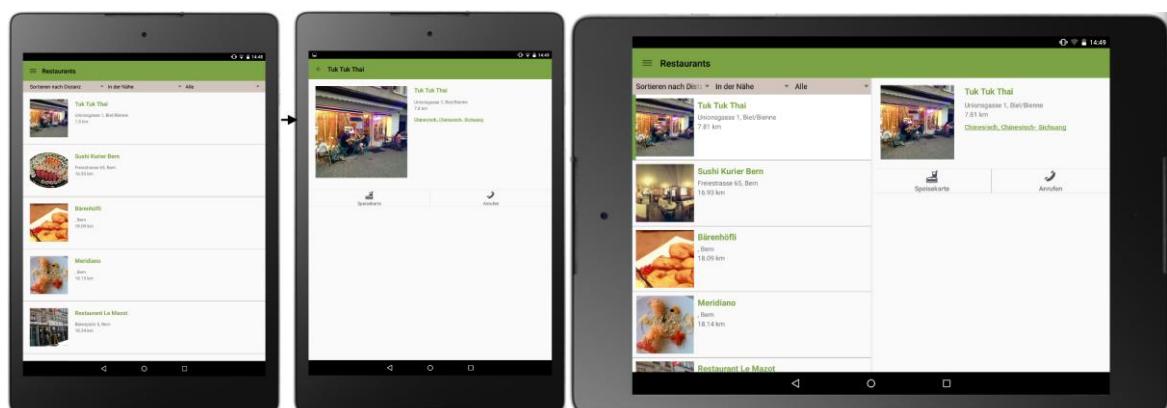


Abbildung 39 Quer- und Hochformat einer App, welche Fragments verwendet (Nexus 9 Tablet)

Eine andere Möglichkeit ist, die Abstände zwischen den GUI-Elementen im Querformat so zu verändern, dass nicht zu viele konzentrierte Leerflächen vorhanden sind.

Für beide Möglichkeiten ist es notwendig, ausrichtungsspezifische Layout-Files für Fragments und Activities zu erstellen.

Ausrichtungsspezifische Layout-Files erstellen

Erstellen Sie im res-Verzeichnis ein Verzeichnis „layout-land“. In diesem Verzeichnis legen Sie für jede Activity und für jedes Fragment, welches im Querformat anders angezeigt werden soll als im Hochformat, eine XML-Layout-Datei für das angepasste Layout.

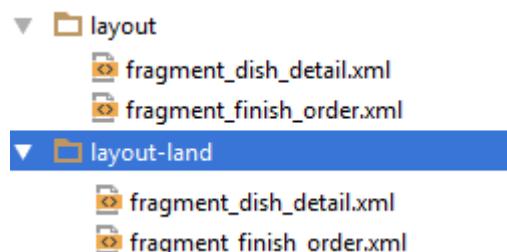


Abbildung 40 Ausrichtungsspezifische Layout-Files

In den Landscape XML-Layout-Dateien können Sie dann entweder andere Dimensionen für die Abstände angeben oder das GUI komplett umstrukturieren. Wenn Sie im Landscape-Modus zwei Fragments in einer Activity anzeigen möchten (wie in der Beispiel-App vom Bild oben), so müssen Sie jeweils eine Layout-Datei der Activity im normalen und im layout-land-Verzeichnis hinterlegen.

Mehrere Fragments in einer Activity anzeigen

Nachfolgende Codeausschnitte zeigen den Code einer Activity (Layout-File) im Portrait-Modus (horizontal) und im Landscape-Modus. Im Landscape-Modus können 2 Fragments in der Activity angezeigt werden. Die folgenden Codeausschnitte stammt von:

<http://developer.android.com/guide/components/fragments.html#Example>

Portrait-Modus	Landscape-Modus
<pre><FrameLayout xmlns:android="http://schemas.android.com/apk/res/android" android:layout_width="match_parent" android:layout_height="match_parent"> <fragment class="com.example.android.apis.app.FragmentLayout\$TitlesFragment" android:id="@+id/titles" android:layout_width="match_parent" android:layout_height="match_parent" /> </FrameLayout></pre>	<pre><LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:orientation="horizontal" android:layout_width="match_parent" android:layout_height="match_parent"> <fragment class="com.example.android.apis.app.FragmentLayout\$TitlesFragment" android:id="@+id/titles" android:layout_weight="1" android:layout_width="0px" android:layout_height="match_parent" /> <FrameLayout android:id="@+id/details" android:layout_weight="1" android:layout_width="0px" android:layout_height="match_parent" android:background="?android:attr/detailsElementBackground" /> </LinearLayout></pre>

Abbildung 41 Gegenüberstellung Code 1 Fragment und 2 Fragment in einer Activity anzeigen

Zu sehen ist, dass beide Codeausschnitte den Block `<fragment...>` beinhalten. Damit wird das Fragment für die Listenansicht eingebunden (TitlesFragment). Im Landscape-Modus wird die Listenansicht auch eingebunden. Zudem wird ein FrameLayout eingebunden, in welchem die Detailansicht angezeigt wird. Die Detailansicht wird dynamisch, je nach gewähltem Eintrag in der Listen-Ansicht, aus dem Java-Code in den `FrameLayout`-Container eingefügt.

Sowohl die Listenansicht als auch die Detailansicht nehmen die Hälfte der Bildschirmbreite ein. Dies ist dadurch gegeben, dass das Attribut „`android:layout_width`“ beim Fragment und auch

beim FrameLayout 1 beträgt, also gleich gross ist. Hätte das Fragment die Zahl 1 und das FrameLayout 2, so wäre die Detailansicht doppelt so breit wie die Listenansicht, also 33.3% für die Liste und 66,6% für die Detailansicht.

6.3.3 Views individuell stylen

Es kann vorkommen, dass die Style-Möglichkeiten der Standard GUI-Komponenten vom Android-Framework (Widgets, TextFields, etc.) den grafischen Ansprüchen nicht genügen. Ist das der Fall, so können eigene Drawables erstellt werden, welche das Look & Feel einer Standard-GUI-Komponente verändern.

Diese XML-Files werden im Verzeichnis „res/drawable“ hinterlegt.

Beispiel 1: Image Button erstellen

Das Ziel ist folgender Button: 

1. Erstellen Sie im res/drawable-Verzeichnis ein Drawable-File. Dieses trägt den Namen „button_camera.xml“ und hat folgenden Inhalt:

```
<?xml version="1.0" encoding="utf-8" ?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/ic_camera_button_focused"
          android:state_pressed="true"/>
    <item android:drawable="@drawable/ic_camera_button_focused"
          android:state_focused="true"/>
    <item android:drawable="@drawable/ic_camera_button_normal"/>
</selector>
```

Selector: Ein Selector wird angewendet, wenn das GUI-Element mehrere Status annehmen kann. Hier kann der Button beispielsweise gedrückt werden, fokussiert sein oder sich im Normal-Status befinden.

Item: Die verschiedenen Status des Buttons. Für jeden Status wird ein anderes Bild (befinden sich in den drawable-... Verzeichnissen) hinterlegt. Denn wird der Button fokussiert oder gedrückt, soll der Button dunkler erscheinen.

2. Nun muss beim Button-Element in der Ansicht (Layout-File des Fragments) das erstellte Drawable verlinkt werden und der Image Button wurde erstellt.

```
<Button
    android:id="@+id/button_beobachtung_new_takephoto"
    android:layout_width="@dimen/width_image_button"
    android:layout_height="@dimen/height_image_button"
    android:layout_gravity="center_vertical"
    android:layout_marginLeft="@dimen/spacing_take_photo_button_left"
    android:background="@drawable/button_camera"
/>
```

Beispiel 2: XelHa-farbige Buttons mit runden Rändern erstellen

Das Ziel ist folgender Button:

Beobachtung senden

1. Drawable-File button_green_round.xml erstellen.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >
    <corners
        android:radius="3dp"
    />
    <solid
        android:color="@color/xelhablue"
    />
</shape>
```

2. Button-Element im Layout-File mit Drawable verlinken.

```
    android:background="@drawable/button_green_round"
```

weitere Beispiele:

Viele Beispiele für das stylen von GUI-Komponenten finden Sie im Internet (Google).

Wichtig ist einfach, dass Konzept mit den drawables zu kennen, d.h. zu wissen wo das drawable-File abgelegt und wie dieses verlinkt werden muss.

7 Dynamische Benutzeroberflächen

Bisher haben Sie gesehen, wie mit XML Benutzeroberflächen (Activity & Fragments) erstellt werden können. Wie bereits mehrfach erwähnt, gehört zu jedem XML-File einer Activity oder Fragments eine Java-Klasse. Diese ist verantwortlich wenn das GUI vor der Anzeige mit Daten gefüllt werden muss oder verarbeitet Benutzeraktionen, wie beispielsweise einen Button-Klick.

In diesem Kapitel lernen Sie zu allererst den Activity und Fragment Lifecycle kennen. Hier geht es darum, dass eine Activity und ein Fragment verschiedenen Status haben können, in welchen Sie sich befinden. Eine Activity kann z.B. angezeigt werden, sich im Back Stack befinden oder noch weitere Status annehmen.

Danach lernen Sie, wie Sie Benutzereingaben aus dem GUI in der Java-Klasse verarbeiten können, z.B. wenn die Textfelder in einer Ansicht ausgefüllt und ein Button zum Absenden der ausgefüllten Daten geklickt wurde.

Dann folgen noch wichtige Inputs zum Wechsel von Ansichten. Wie kann Ansicht 1 z.B. Ansicht 2 starten und dieser Daten übergeben.

Zu guter Letzt betrachten wir noch einige wichtige Elemente wie Dialoge, Dropdowns, etc.

7.1 Activity Lifecycle

Die nachfolgende Grafik zeigt den Activity Lifecycle:

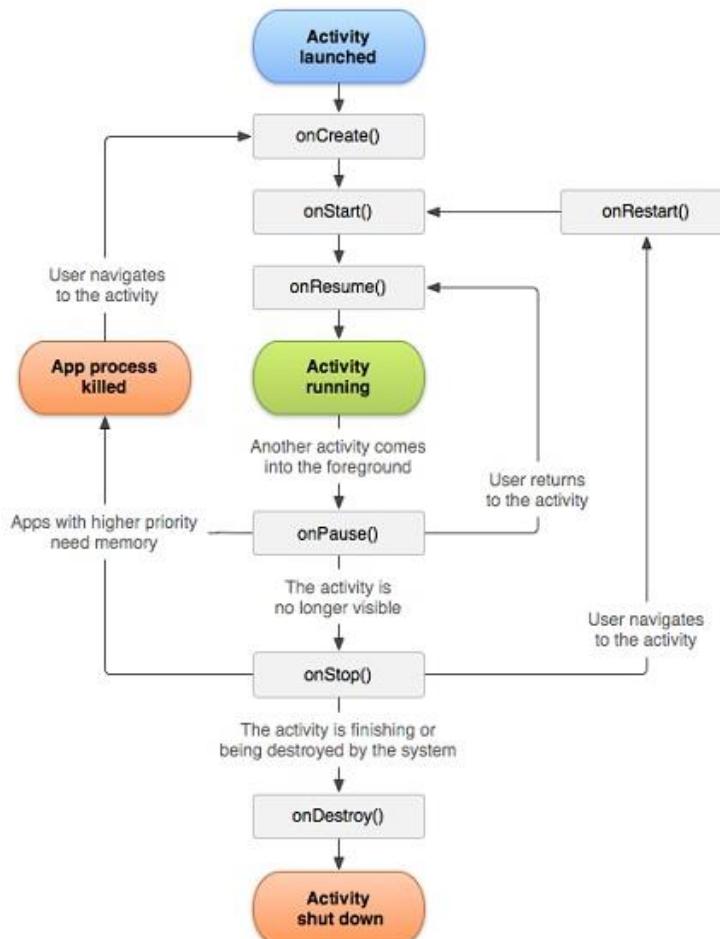


Abbildung 42 Activity Lifecycle [12]

Nachfolgend möchte ich anhand dem Start der App „HelloWorld“ die wichtigsten Punkte des Activity Lifecycle erklären.

1. Android-Benutzer startet die App „HelloWorld“.
2. Die App startet die MainActivity (**Activity launched**). Beim Start einer Activity werden hintereinander folgende Methoden der Java-Klasse der Activity (MainActivity.java) aufgerufen.
 - a. **onCreate()**-Methode
 - b. **onStart()**-Methode
 - c. **onResume()**-Methode

Diese 3 Methoden werden ausgeführt bevor der Benutzer die Ansicht sieht. In der **onCreate()**-Methode soll das GUI aufbereitet werden, d.h. das Layout-File gesetzt oder die GUI-Elemente mit Java-Variablen verbunden werden.

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

onStart() und **onResume()** müssen nicht zwingend vorhanden sein.

3. Die Activity wird nun angezeigt.
4. Der Benutzer startet nun entweder über die App eine neue Activity (z.B. indem er einen Button klickt) oder minimiert die App mit dem Home-Button seines Smartphones.
5. Die Activity wird nun pausiert. Die Methode **onPause()** vom MainActivity wird aufgerufen. Die Activity pausiert.
Wenn die Activity weiterhin im Hintergrund der gestarteten Activity läuft, so bleibt der Status bei „Pausiert“. Dies passiert z.B. wenn mit dem Home-Button eine App minimiert wird. Falls die Activity nicht weiter sichtbar ist (auch nicht im Hintergrund) wird **onStop()** aufgerufen. Dies passiert normalerweise wenn in einer App eine neue Activity gestartet und die alte Activity auf den Backstack gelegt wird.
6. Sobald die neue Activity wieder beendet wird, werden die Methoden **onRestart()**, **onStart()** und **onResume()** des alten Activity aufgerufen und die Activity wieder angezeigt.

Die wichtigste Methode innerhalb einer Activity-Klasse ist **onCreate()**. Diese muss zwingend in jeder Activity-Klasse vorhanden sein um das GUI aufzubereiten. Alle anderen Methoden sind freiwillig. Ab und zu wird auch **onPause()** implementiert, um vor dem pausieren einer Ansicht dessen aktueller Status zwischen zu speichern, damit dieser dann in **onResume()** wiederhergestellt werden kann.

7.2 Fragment Lifecycle

Auch Fragments haben einen Lebenszyklus. Die Methoden `onAttach()`, `onCreate()` etc. werden (müssen) in der Java-Klasse des Fragments implementiert.

Das Betriebssystem ruft beim Start einer Ansicht immer die Java-Klasse des Fragments auf und führt die Methoden `onAttach()`, `onCreate()`, usw. (siehe Abbildung) der Java-Klasse in einer bestimmten Reihenfolge aus.

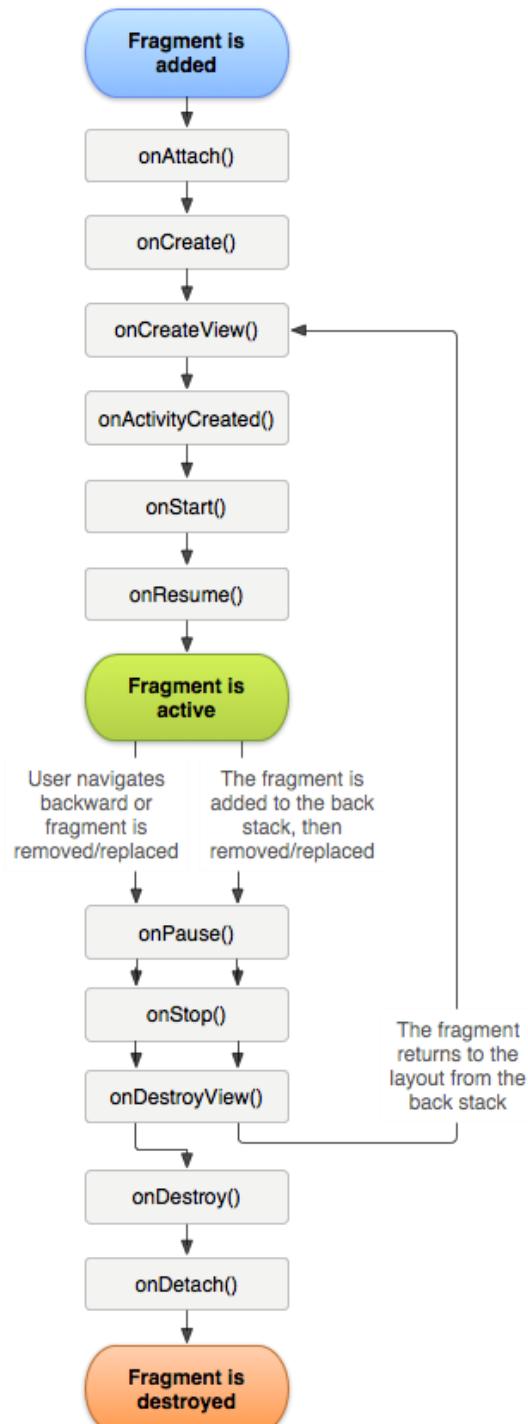


Abbildung 43 Fragment Lifecycle

Wichtig ist, dass Sie mindestens die 2 folgenden Methoden in der Fragment-Klasse hinzufügen:

- **onCreate():**

Wird aufgerufen, wenn das System das Fragment erstellt. Diese Methode wird nur bei der Erstellung des Fragments aufgerufen, nicht jedoch wenn das Fragment wiederhergestellt wird, nachdem es pausiert wurde. Daher sollten in dieser Methode alle Elemente des Fragments initialisiert werden, welche bei einer Wiederherstellung des Fragments nicht neu initialisiert werden sollen. Dies können z.B. GUI-Elemente sein, welche immer denselben Inhalt anzeigen. Keinen Sinn macht es aber GUI-Elemente wie Listen, wessen Inhalte dynamisch sind, im onCreate zu initialisiere. Denn es könnte ja sein, dass sich die Inhalte geändert haben, während das Fragment pausierte (z.B. weil DB mit den Daten von extern geändert wurde oder weil die Ansicht pausiert wurde, damit in einer anderen Ansicht Daten hinzugefügt werden konnten).

- **onCreateView():**

Wird aufgerufen, wenn das System das User Interface des Fragments zum ersten Mal zeichnet. Wird im Gegensatz zu onCreate() erneut aufgerufen, wenn ein pausiertes Fragment aus dem Backstack wieder angezeigt werden soll.

In dieser Methode werden typischerweise die GUI-Elemente initialisiert, d.h. die GUI-Elemente aus dem XML-File an Java-Objekte zugeordnet, damit z.B. Button-Klicks abgefangen werden können (weitere Infos folgen), Spinners (Dropdowns) mit Daten gefüllt, andere GUI-Elemente mit Daten gefüllt, OnClick-Listeners definiert, etc.

Diese Methode ist die wichtigste Methode des Fragments. Zusammengefasst kann gesagt werden, dass in dieser Methode das GUI aufbereitet wird.

Eine weitere wichtige Methode ist:

- **onPause():**

Wird aufgerufen, wenn der User das Fragment verlässt, z.B. weil ein anderes Fragment aufgerufen wird oder wenn die App minimiert wird. In dieser Methode sollte alles zwischengespeichert werden, was bei einer Wiederherstellung der Ansicht wiederhergestellt werden soll (Status des GUIs, z.B. ausgewählter Dropdown-Eintrag, etc.).

Die anderen Methoden werden eher selten implementiert. Weitere Informationen zum Fragment Lifecycle finden Sie auf folgender Webseite:

<https://developer.android.com/guide/components/fragments.html#Creating>

7.3 Activity-XML mit Java-Klasse verbinden

Wie bereits mehrfach erläutert besteht eine Ansicht, welche mit einer Activity realisiert wurde, immer aus einem XML-File und einer Java-Klasse.

Das System ruft beim Start einer Ansicht immer die Java-Klasse, d.h. deren Methoden des Lifecycles auf. Aufgerufen wird unter anderem die Methode „onCreate()“. In dieser Methode muss das Layout, d.h. das XML-File der Activity angehängt werden.

Nachfolgender Codeausschnitt zeigt, wie das Layout (XML-Datei) der Activity mit der Java-Klasse verbunden wird. Bei Android spricht man von „inflate“, Englisch für „aufblasen“.

```
public class MainActivity extends AppCompatActivity {

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

Relevant ist die Zeile `setContentView(R.layout.activity_main);`

Hier wird das Layout „activity_main.xml“ verbunden. In diesem Codeausschnitt wird das Layout nur geladen. Das Activity wird also 1:1 so angezeigt, wie im Layout-File (XML) definiert. Es werden keine GUI-Elemente im Programmcode initialisiert, d.h. z.B. mit Daten gefüllt.

Um GUI-Elemente (Widgets, Text Fields, etc.) im Java-Code anzusprechen um diese z.B. mit Daten zu füllen oder ActionListener hinzuzufügen um z.B. Button-Klicks abzufangen, ist es notwendig, in `onCreate()` das GUI-Element einer Java-Variable zuzuteilen.

Nachfolgendes Beispiel zeigt, wie ein Button aus dem Layout-File einer Variablen zugewiesen wird um diesem einen ActionListener hinzuzufügen.

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

    Button takePhotoButton = findViewById(R.id
        .button_beobachtung_new_takephoto);
    takePhotoButton.setOnClickListener(mTakePhotoOnClickListener);

    return rootView;
}
```

Im nächsten Kapitel wird die Verarbeitung von Button-Klicks im Detail angeschaut.

7.4 Benutzereingaben verarbeiten

Nachfolgend wird Ihnen Schritt für Schritt aufgezeigt, wie ein kleines Formular mit einem Text-Feld zur Eingabe eines Namens und einem Speichern-Button erstellt wird. Der Benutzer gibt im Textfeld einen Namen ein und klickt dann auf den Speichern-Button. Nun wird der eingegebene Name in einem Toast auf dem Bildschirm angezeigt.

Anhand dieses Beispiels sehen Sie einerseits, wie Benutzereingaben wie Texte im Java-Code verarbeitet, andererseits wie Aktionen, wie z.B. einen Klick auf einen Button, im Java-Code behandelt werden.

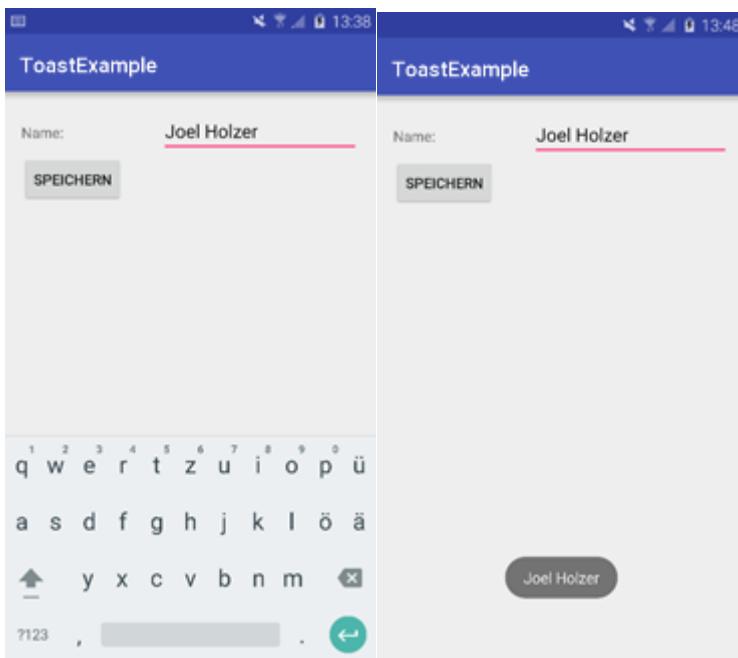


Abbildung 44 Demo-App zum Verarbeiten von Benutzereingaben (ActionListener)

Der Code des folgenden Beispielprojektes befindet sich auf dem **GitHub-Repository** von Joel Holzer unter dem Namen **ToastExample**.

7.4.1 Schritt 1: Activity/Fragment erstellen und Layout-File umsetzen

1. Erstellen Sie das Activity oder Fragment.
2. Setzen Sie das Layout-File (XML) des Activities/Fragments um indem Sie im Designer und im Text-Editor folgende GUI-Elemente platzieren:
 - a. Plain Text View für das Label „Name“.
 - b. Plain Text Field zur Eingabe des Namens.
 - c. Button „Speichern“ zur Ausgabe des Namens.

Der Code des Layout-Files (XML) ist dann wie folgt:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="ch.nyp.toastexample.MainActivity"
    android:orientation="vertical"
    android:layout_margin="@dimen/activity_horizontal_margin"
    >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/main.textview.name"
            />

        <EditText
            android:id="@+id/edittext_main_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/main.edittext.name"
            />

    </LinearLayout>

    <Button
        android:id="@+id/button_main_save"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/main.button.save"
        />

</LinearLayout>
```

Wichtig ist, dass alle GUI-Elemente, welche vom Java-Code aus angesprochen werden, eine ID haben müssen. Dies ist der Speichern-Button, wessen Klick abgefangen werden muss, sowie das Textfeld (EditText) zur Eingabe des Namens. Bei diesem muss der eingegebene Name ausgelesen werden können.

Die ID eines GUI-Elements muss innerhalb des gesamten Projekts eindeutig sein.

Empfehlung: Wählen Sie die ID immer so, dass anhand der ID erkannt werden kann, um was für einen Typ von GUI-Element es sich handelt (z.B. Button, EditText, etc.), in welchem Activity/Fragment sich das GUI-Element befindet und um welches GUI-Element.

Beispiel: edittext_main_name → Es handelt sich um ein EditText-Feld, in dem Activity/Fragment „main“ und das Element selbst ist für die Eingabe des Namens, darum „name“.

7.4.2 Schritt 2: GUI-Elemente im Java-Code einer Variable zuteilen

Damit die Eingabe aus dem Text-Feld (EditText) und der Button-Klick abgefangen werden können, müssen in einem ersten Schritt die GUI-Elemente aus dem XML-File an eine Java-Variable gekoppelt werden.

Falls Sie mit Activities arbeiten (wie im abgegebenen Beispiel-Code):

Dies geschieht in der Methode „**onCreate**“ der Java-Klasse mit den beiden Zeilen. „mEditTextName“ und mButtonSave“ werden nun mit dem GUI-Element aus dem Layout-File verbunden.

```
mEditTextName = (EditText) rootView.findViewById(R.id.edittext_addname_name);  
mButtonSave = (Button) rootView.findViewById(R.id.button_addname_save);
```

Über diese Variablen können später Daten aus dem GUI ermittelt oder Daten zur Anzeige im GUI geschrieben werden. Nachfolgend folgt der Code der ganzen Activity-Klasse. Der Präfix m vor dem Variablenname steht für Member. In Android ist es üblich, bei allen Instanzvariablen den Präfix „m“ davorzusetzen.

```
public class MainActivity extends AppCompatActivity {  
  
    private EditText mEditTextName;  
    private Button mButtonSave;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        mEditTextName = findViewById(R.id.edittext_main_name);  
        mButtonSave = findViewById(R.id.button_main_save);  
    }  
}
```

Falls Sie mit Fragments arbeiten:

Dies geschieht in der Methode „**onCreateView**“ der Java-Klasse mit den beiden Zeilen. „mEditTextName“ und mButtonSave“ werden nun mit dem GUI-Element aus dem Layout-File verbunden.

```
mEditTextName = (EditText) rootView.findViewById(R.id.edittext_addname_name);  
mButtonSave = (Button) rootView.findViewById(R.id.button_addname_save);
```

Über diese Variablen können später Daten aus dem GUI ermittelt oder Daten zur Anzeige im GUI geschrieben werden. Nachfolgend folgt der Code der ganzen Fragment-Klasse. Der Präfix m vor dem Variablenname steht für Member. In Android ist es üblich, bei allen Instanzvariablen den Präfix „m“ davorzusetzen.

```
public class AddNameFragment extends Fragment {  
  
    private EditText mEditTextName;  
    private Button mButtonSave;  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
        //Layout (XML) inflate  
        View rootView = inflater.inflate(R.layout.fragment_addname, container, false);  
  
        //EditText und Button aus XML einer Variable zuteilen  
        mEditTextName = rootView.findViewById(R.id.edittext_addname_name);  
        mButtonSave = rootView.findViewById(R.id.button_addname_save);  
  
        return rootView;  
    }  
}
```

7.4.3 Schritt 3: OnClickListener für Button-Klick erstellen und der Button-Variable zuweisen

Aktionen werden bei Android wie bei Java üblich mit ActionListener abgefangen/behandelt.

Oben in der Activity oder Fragment-Klasse kann nun ein OnClickListener für den Save-Button erstellt werden:

```
private View.OnClickListener mSaveOnClickListener = new View.OnClickListener() {
    @Override
    public void onClick(View sendButton) {
        //Aktion, welche beim Button-Klick ausgeführt werden soll.
    }
};
```

Dieser OnClickListener muss nun in der onCreate-Methode (bei Activity), oder in der onCreateView-Methode beim Fragment, dem Button hinzugefügt werden:

```
mButtonSave.setOnClickListener(mSaveOnClickListener);
```

Der Code der ganzen Java-Klasse sieht beim **Activity** nun wie folgt aus:

```
public class MainActivity extends AppCompatActivity {

    private EditText mEditTextName;
    private Button mButtonSave;

    private View.OnClickListener mSaveOnClickListener = new View.OnClickListener() {
        @Override
        public void onClick(View sendButton) {
            //Aktion, welche beim Button-Klick ausgeführt werden soll.
        }
    };

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

        mEditTextName = findViewById(R.id.edittext_main_name);
        mButtonSave = findViewById(R.id.button_main_save);

        mButtonSave.setOnClickListener(mSaveOnClickListener);
    }
}
```

Beim **Fragment** würde der Code wie folgt aussehen:

```
public class AddNameFragment extends Fragment {

    private EditText mEditTextName;
    private Button mButtonSave;

    private View.OnClickListener mSaveOnClickListener = new View.OnClickListener() {
        @Override
        public void onClick(View sendButton) {
            //Aktion, welche beim Button-Klick ausgeführt werden soll.
        }
    };

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        //Layout (XML) inflate
        View rootView = inflater.inflate(R.layout.fragment_addname, container, false);

        //EditText und Button aus XML einer Variable zuteilen
        mEditTextName = rootView.findViewById(R.id.edittext_addname_name);
        mButtonSave = rootView.findViewById(R.id.button_addname_save);
        mButtonSave.setOnClickListener(mSaveOnClickListener);

        return rootView;
    }
}
```

7.4.4 Schritt 4: eingegebener Name in Toast ausgeben

Nun muss noch der Inhalt der onClick-Methode des OnClickListeners programmiert werden, d.h. die Aktion, welche beim Button-Klick ausgeführt wird. In unserem Fall soll der Inhalt vom EditText (Text-Feld) ausgelesen und in einem Toast ausgegeben werden.

Der Code des OnClickListener des Buttons sieht dann wie folgt aus:

```
private View.OnClickListener mSaveOnClickListener = new View.OnClickListener() {
    @Override
    public void onClick(View sendButton) {
        //Eingegebener Text in einem Toast, welches 3.5 Sekunden (Toast.LENGTH_LONG)
        angezeigt
        // wird, ausgeben
        String name = mEditTextName.getText().toString();
        Toast toast = Toast.makeText(getApplicationContext(), name, Toast.LENGTH_LONG);
        toast.show();

    }
};
```

Zuerst wird der eingegebene Text der Variable „name“ zugewiesen und die Variable dann in einem Toast angezeigt. Ein Toast ist ein kleines Popup, welches für eine bestimmte Zeit auf dem Bildschirm angezeigt wird. In unserem Fall wird das Toast 3.5 Sekunden (Toast.LENGTH_LONG) angezeigt.

Das erläuterte Prinzip gilt für alle Arten von GUI-Elementen. Um auf GUI-Elemente zuzugreifen, d.h. gemachte Angaben vom User auszulesen oder das GUI mit Daten zu füllen oder Aktionen abzufangen, muss als erster Schritt immer eine Variable für das GUI-Element erstellt und mit dem GUI-Element aus dem XML-File verbunden werden.

Alle Arten von ActionListener können mit set.... gefolgt vom entsprechenden Listener zum GUI-Element hinzugefügt werden. Das gilt nicht nur für OnClickListener.

7.4.5 Schritt 5, falls mit Fragment gearbeitet wird: Fragments zum Activity hinzufügen

Fragments werden in Activities reingeladen, entweder ein Fragment pro Activity oder mehrere Fragments pro Activity. Jede App braucht aber auch beim Einsatz von Fragments zwingend eine Activity, die sogenannte Main-Activity.

Es gibt zwei Arten wie ein Fragment in ein Activity reingeladen werden kann:

Fragments per XML zum Activity hinzufügen:

<https://developer.android.com/training/basics/fragments/creating.html>

Fragmente zur Laufzeit zum Activity hinzufügen (im Java Code):

<https://developer.android.com/training/basics/fragments/fragment-ui.html>

7.5 Neue Ansicht starten

Bisher haben Sie gehört, wie Ansichten mit Activities und Fragments erstellt und GUI-Eingaben im Java-Code verarbeitet werden können. Die von Ihnen erstellte Activities und Fragments sind jedoch bisher auf sich alleine gestellt, d.h. führen nur Aktionen aus, welche die eigene Activity oder das eigene Fragment betreffen.

Nachfolgend wird Ihnen erklärt, wie Sie in einer App eine neue Ansicht starten (Ansicht wechseln) und Daten von einer Ansicht zur anderen übergeben können. Dies passiert mit Intents.

7.5.1 Was ist ein Intent?

Um eine neue Activity zu starten und dieser Activity Daten zu übergeben, werden Intents (Deutsch Absichten) benötigt.

Es gibt 2 Arten von Intents:

- **Explizite Intents:** aktivieren/starten eine bekannte Komponente, z.B. eine Activity in der eigenen App oder eine Activity einer spezifischen anderen App.
- **Implizite Intents:** führen eine Aktion aus, ohne die Komponente, welche die Aktion verarbeitet, zu kennen. Als Beispiel können wir eine App nehmen, welche einen Link auf eine Webseite beinhaltet. Klickt der Benutzer in der App auf den Link, so wird ein impliziter Intent für alle Browser-Apps erstellt. Ist auf dem mobilen Gerät nur ein Browser installiert, so wird der Link automatisch in diesem Browser geöffnet. Sind mehrere Browser installiert, so muss der Benutzer in einem Dialog auswählen, in welcher App der Link geöffnet werden soll.

Wir konzentrieren uns in dieser Dokumentation auf explizite Intents.

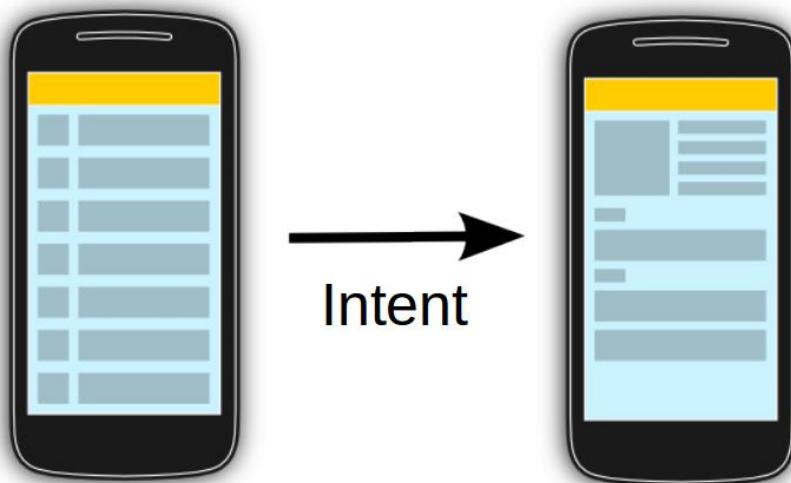


Abbildung 45 Expliziter Intent, welcher eine neue Activity startet [13]

7.5.2 Intent zum Starten einer Activity

Gegeben ist folgende Ausgangslage:

- MainActivity
- Activity2

Das MainActivity beinhaltet ein Text-Feld zur Eingabe eines Textes und den Button „Activity 2 öffnen“. Klickt der Benutzer auf den Button, so soll Activity2 aufgerufen werden. In diesem Activity wird dann der eingegebene Text ausgegeben.

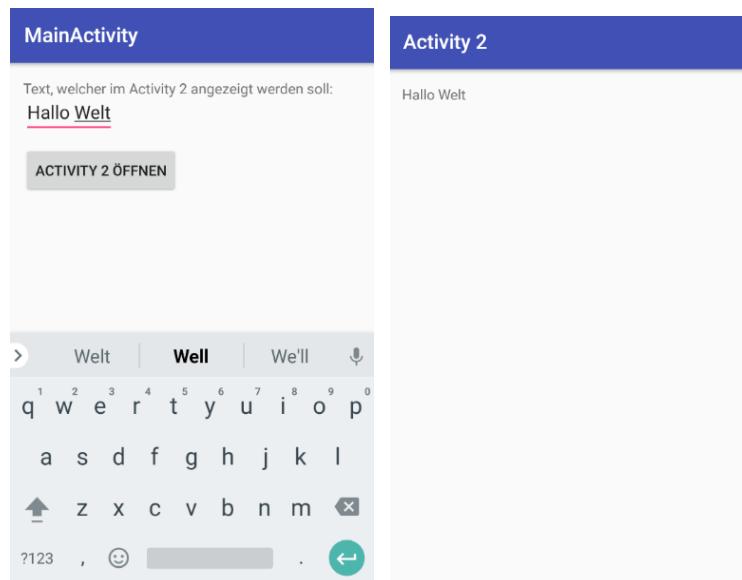


Abbildung 46 Demo App zum Starten einer neuen Activity mit einem Intent

Der Code des folgenden Beispielprojektes befindet sich auf dem **GitHub-Repository** von Joel Holzer unter dem Namen **IntentDemo**.

Variante 1: Activity starten ohne Datenübergabe

Nachfolgender Code des Button-OnClickListeners startet das Activity „ShowNameActivity“.

```
private View.OnClickListener mOpenActivityOnClickListener = new View.OnClickListener()
{
    @Override
    public void onClick(View sendButton) {
        //Activity2 starten
        startActivity(new Intent(getActivity(), Activity2.class));
    }
};
```

Relevant ist die Zeile `startActivity(new Intent(getActivity(), Activity2.class));`

- `startActivity()`: startet eine neue Activity über den Intent
- `getActivity()`: der erste Parameter des Intents ist die Activity, von welcher der Intent ausgeht (die Activity, welche im Moment angezeigt wird)
- `Activity2.class`: Activity welche gestartet werden soll.

Variante 2: Activity starten mit Datenübergabe

Um Daten von einer Activity an die nächste zu übergeben, müssen dem Intent, welcher die neue Activity startet sogenannte „Extras“ übergeben werden. Dies geht mit dem Keyword „**putExtra**“.

Nachfolgender Code zeigt die Übergabe des eingegebenen Namens an die **ShowNameActivity**:

```

private View.OnClickListener mSaveOnClickListener = new View.OnClickListener() {
    @Override
    public void onClick(View sendButton) {
        //ShowNameActivity starten und Name in diesem Activity darstellen
        String name = mEditTextName.getText().toString();

        Intent showNameActivityIntent = new Intent(getApplicationContext(),
                                                       ShowNameActivity.class);
        showNameActivityIntent.putExtra("key_person_name", name);
        startActivity(showNameActivityIntent);
    }
};

private View.OnClickListener mOpenActivityOnClickListener = new View.OnClickListener()
{
    @Override
    public void onClick(View openActivityButton) {
        String inputText = mTextToTransferEditText.getText().toString();

        Intent intent = new Intent(getApplicationContext(), Activity2.class);
        Bundle bundle = new Bundle();
        bundle.putString(INTENT_KEY_DISPLAY_TEXT_ACTIVITY_2, inputText);
        intent.putExtras(bundle);
        startActivity(intent);
    }
};
  
```

Relevant sind folgende Zeilen:

```

Intent intent = new Intent(getApplicationContext(),Activity2.class);
Bundle bundle = new Bundle();
bundle.putString("key_text", inputText);
intent.putExtras(bundle);
startActivity(intent);
  
```

Zuerst wird der **Intent** zum Aufruf der neuen Ansicht als Objekt vom Typ „Intent“ erstellt.

Danach wird diesem Intent ein **Key-Value-Pair mit Daten** angehängt. In unserem Fall wird für den Key „key_text“ der eingegebene Text übergeben. Jeder Key muss eindeutig sein. Dieser Key wird benötigt, um die Daten in der neuen (gestarteten) Activity zu holen.

Zu guter Letzt wird der Intent der **startActivity**-Methode übergeben, damit die Activity gestartet wird.

Nun müssen noch das Activity2 erweitert werden, damit die übergebenen Daten auf dem Bildschirm angezeigt werden. Das ShowNameActivity muss die Daten vom Intent entgegennehmen und in einer TextView anzeigen.

Nachfolgend der Code der Activity2:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main2);  
    setTitle(R.string.activity2_title);  
  
    Bundle intentBundle = this.getIntent().getExtras();  
  
    TextView transferredTextView = this.findViewById(R.id.textView_2_transferredText);  
  
    transferredTextView.setText(intentBundle.getString(MainActivity.INTENT_KEY_DISPLAY_TEXT  
    _ACTIVITY_2));  
}
```

7.5.3 Impliziter Intent zum Starten der Kamera

Unter nachfolgendem Link ist beschrieben, wie mit Hilfe eines impliziten Intents die Kamera-App gestartet wird und dann das geschossene Foto an die App zurückgegeben wird:

<https://developer.android.com/training/camera/photobasics.html>

7.6 Dialoge

Dialoge gibt es nicht nur in Java-Swing-Anwendungen, sondern auch bei Android-Apps. Sie werden vor allem benutzt, um vom Benutzer eine Bestätigung vor der Ausführung von bestimmten Aktionen (z.B. Löschen von Daten) einzuholen oder den Benutzer über Aktionen, welche im Hintergrund ausgeführt werden oder ausgeführt wurden, zu informieren.

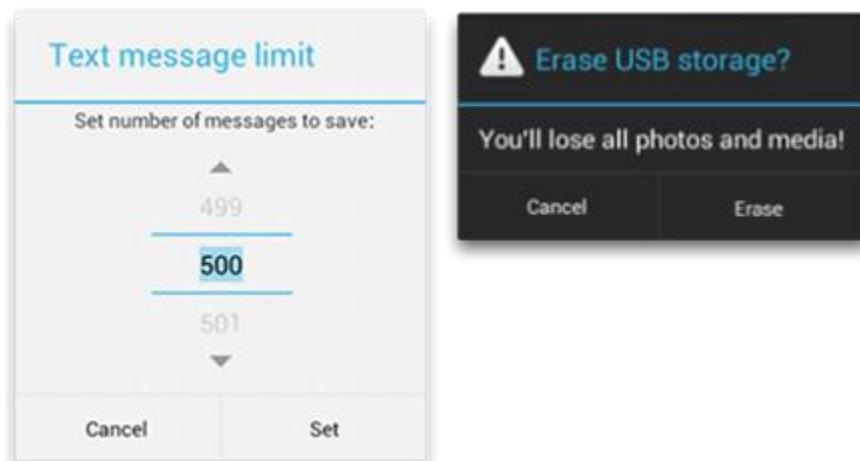


Abbildung 47 Beispiele von Dialogen [14]

7.6.1 Dialog erstellen

Unter nachfolgendem Link wird gezeigt, wie folgender Dialog (und auch andere Dialoge) erstellt werden kann:

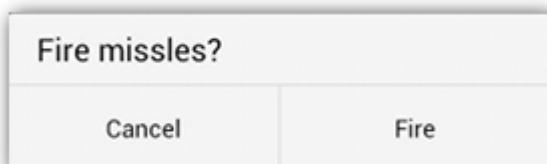


Abbildung 48 Dialog, welcher erstellt wird

<https://developer.android.com/guide/topics/ui/dialogs.html#DialogFragment>

Die wichtigste Klasse zur Erstellung von Dialogen ist die Klasse „AlertDialog“. Ein Alert-Dialog kann über einen oder über zwei Buttons verfügen (meistens über Zwei). Einer dieser Button ist der positive-Button (z.B. Ja), der andere der negative Button (z.B. Nein). Die Aktion der beiden Buttons und auch der Text des Buttons können vom Entwickler definiert werden. Es wäre also auch möglich, den positiven-Button mit dem Text „Nein“ und der Abbrechen-Aktion auszustatten. Dies macht jedoch nicht wirklich Sinn.

Der positive Button ist immer der rechte Button, der negative Button der Linke.

7.7 Spinners (Dropdowns)

Bei Android wird ein Dropdown „Spinner“ genannt. Ein Spinner kann entweder statische, fixe Einträge oder einen dynamischen Inhalt (z.B. von einem Webservice oder aus der Datenbank) beinhalten.

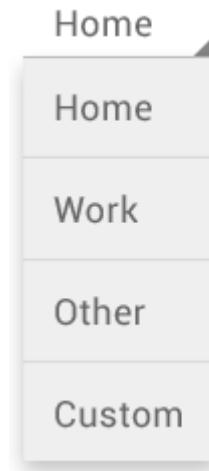


Abbildung 49 Spinner [15]

7.7.1 Spinner mit statischen Einträgen erstellen

Um einen Spinner mit statischen, fixen Einträgen zu erstellen (z.B. mit den Einträgen Home, Work, Other, Custom) müssen Sie wie folgt vorgehen:

1. Platzieren Sie in dem Activity/Fragment wo der Spinner angezeigt werden soll ein **Spinner-Widget** und geben Sie diesem eine ID (um später den Spinner mit Daten abzufüllen).
2. Da die Spinner-Einträge fix sind, werden diese in einem String-File definiert. Das passiert jedoch nicht wie die Definition der normalen Texte in der Datei „res/values/strings.xml“, sondern in der Datei „**res/values/arrays.xml**“. Diese Datei muss erstellt werden, falls sie noch nicht vorhanden ist.

In dieser Array-Datei wird nun ein Array mit allen Einträgen des Spinners (Home, Work, Other, Custom) definiert:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="main.spinner.places">
    <item>Home</item>
    <item>Work</item>
    <item>Other</item>
    <item>Custom</item>
  </string-array>
</resources>
```

Unter dem Namen „showname.spinner.places“ kann das Array aufgerufen/geholt werden. Dieser muss einmalig sein innerhalb aller String-Files (arrays.xml, strings.xml). Wie bei den String-Files kann auch das arrays.xml in allen Sprachen hinterlegt werden, um die Spinner Einträge zu übersetzen (arrays-de.xml, array-fr.xml, usw.)

3. Nun muss in der **onCreate()**-Methode des Activities (onCreateView bei Fragment) noch der Spinner mit dem Array „spinner_place“ abgefüllt werden, damit die Einträge „Home“, „Work“, etc. dann auch wirklich im Spinner erscheinen und ausgewählt werden können:

```
Spinner placesSpinner = findViewById(R.id.spinner_main_places);
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
    R.array.addname_spinner_places, android.R.layout.simple_spinner_item);
placesSpinner.setAdapter(adapter);
```

Wichtig ist, dass einem Spinner, unabhängig ob dessen Einträge statisch sind oder dynamisch, ein **Adapter** (z.B. ArrayAdapter) hinzugefügt wird. Dieser Adapter wird dann mit Daten gefüllt. In unserem Fall wird der **Array-Adapter** mit einer **CharSequence** gefüllt, d.h. mit dem Array „**main.spinner.places**“ aus dem File **arrays.xml**.

Als weiterer Parameter wird bei uns `android.R.layout.simple_spinner_item` übergeben.

Dies ist das **Layout** für einen Eintrag des Spinners. Mit dem Layout kann definiert werden, wie ein einzelner Eintrag im Spinner aussieht, z.B. die Formatierung der Schrift (Fett, Schriftgrösse, Ausrichtung) etc. In diesem Beispiel wird ein Standard-Layout vom Android-Framework verwendet. Es ist jedoch möglich, eigene Layouts zu erstellen (im Verzeichnis res/layout) um das Look & Feel so zu gestalten, wie es gewünscht wird.

Damit die Einträge des Adapters nun angezeigt werden, muss der Adapter am Schluss noch zum Spinner hinzugefügt werden (**setAdapter**).

Der Code des Beispielprojektes befindet sich auf dem **GitHub-Repository** von Joel Holzer unter dem Namen **StaticSpinnerDemo**.

7.7.2 Spinner-Selektion ermitteln

Unabhängig davon, ob ein Spinner mit statischen oder dynamischen Einträgen erstellt wird, kann der selektierte Eintrag des Spinners mit folgender Codezeile ermittelt werden:

```
mSpinnerPlaces.getSelectedItem()
```

Diese Methode gibt das selektierte Objekt zurück als „Object“. Beim vorherigen-Beispiel mit dem String-Array beinhaltet der Spinner String-Objekte, daher kann **getSelectedItem()** nach String gecastet werden und liefert den selektierten String (z.B. Work, Home, etc.)

Eine andere Variante ist die Position des selektierten Eintrags im Spinner zu ermitteln. Gerade **bei mehrsprachigen Anwendungen** kann der selektierte Eintrag nicht über den selektierten String ermittelt werden, da dieser in jeder Sprache anders ist. In diesem Fall macht es bei Spinners mit String-Array Sinn, über die Position zu gehen:

```
int selectedSpinnerPosition = mSpinnerPlaces.getSelectedItemPosition();
```

Bei Spinnern mit dynamischen Einträgen sollte der selektierte Eintrag jedoch immer mit **getSelectedItem()** ermittelt werden.

7.7.3 Spinner mit dynamischen Einträgen (Objekt-Einträgen) und eigenem Layout erstellen

Nachfolgend soll gezeigt werden, wie ein Spinner erstellt werden kann, welcher eine Liste von Objekten darstellt. Als Beispiel wird der Spinner aus der Demo App „DynamicSpinnerDemo“ erläutert. Dabei werden Lernende mit Profilbild, Name und Vorname angezeigt. Damit ein Spinner-Eintrag nicht als String angezeigt wird, sondern die Anzeige von Foto und String möglich ist, muss ein eigenes Layout erstellt werden.

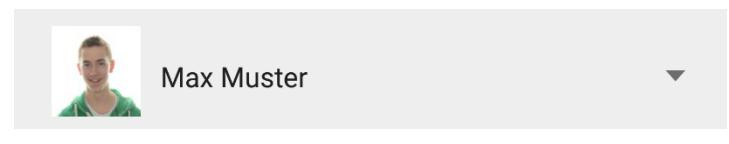


Abbildung 50 Spinner mit dynamischen Einträgen und eigenem Layout

1. Platzieren Sie in dem Activity/Fragment wo der Spinner angezeigt werden soll ein **Spinner-Widget** und geben Sie diesem eine ID (um später den Spinner mit Daten abzufüllen).
2. Erstellen Sie nun die Layout-Datei für das Look & Feel eines einzelnen Eintrags im Spinner. Schauen Sie sich dazu die Datei „res/layout/layout_image_spinner_item_list“ an. Dieses Layout beinhaltet eine ImageView für die Darstellung des Profilfotos, sowie eine TextView für die Anzeige des Namens.
3. Erstellen Sie nun eine eigene Adapter-Klasse für die Darstellung der Lernenden. Diese Klasse muss von der Klasse „ArrayAdapter“ erben. In unserem Fall beinhaltet unser Adapter User-Objekte (Die Model-Klasse für die User-Objekte muss vorgängig erstellt werden).

Schauen Sie sich dazu die Klassen „**LernendeAdapter.java**“ und „**User**“ an. Äusserst wichtig ist der **Konstruktor**. Im Konstruktor vom LernendeAdapter wird eine Liste von User übergeben. Dies sind die User, welche im Spinner angezeigt werden sollen.

Wichtig sind zudem die Methoden **getDropdownView(...)** und **getView(...)**. Diese Methoden erhalten mit dem Parameter „**position**“ die Position des anzuzeigenden Eintrags im Spinner. Sie bereitet dann den Eintrag des Spinners an dieser Position auf und gibt diesen Eintrag zurück. Dazu wird das Layout-File für den Spinner-Eintrag geladen (inflated) und die GUI-Elemente dieses Layouts mit den Daten eines Users gefüllt. Der User, an der übergebenen Position ist immer der User an derselben Position in der dem Konstruktor übergebenen Liste. D.h. wird **getView** die Position 1 übergeben, so wird der Eintrag für den User an der Position 1 in der Liste, die dem Konstruktor übergeben wurde, angezeigt.

getView(...) wird aufgerufen, wenn ein Eintrag im Spinner selektiert ist, d.h. der Spinner nicht aufgeklappt ist. **getDropdownView(...)** wird aufgerufen, wenn der Spinner aufgeklappt wird.

Der LernendeAdapter wurde mit einem **ViewHolder** umgesetzt. ViewHolder ermöglichen, dass das Scrollen in einem Spinner nicht ruckelt. Weitere Infos zu ViewHolder, siehe <https://developer.android.com/training/improving-layouts/smooth-scrolling.html>

4. Zu guter Letzt muss in der **onCreate**-Methode des Activities der Spinner initialisiert und der bei Punkt 3 erstellte Adapter zugewiesen werden (mit **setAdapter(...)**);

Der Code des Beispielprojektes befindet sich auf dem **GitHub-Repository** von Joel Holzer unter dem Namen **DynamicSpinnerDemo**.

7.8 ListView

Nachfolgend soll gezeigt werden, wie eine ListView erstellt werden kann, welche eine Liste von Objekten darstellt. Als Beispiel wird die ListView aus der Demo App „ListViewExample“ erläutert. Dabei werden von 30 Einträgen jeweils eine Nummer und ein Titel angezeigt.

Die Erstellung einer dynamischen ListView ist ziemlich ähnlich wie bei einem Spinner und läuft auch über einen ArrayAdapter.

Nachfolgend wird Schritt für Schritt die Erstellung der genannten ListView erläutert.

ListViewExample	
Random Item	1
Random Item	2
Random Item	3
Random Item	4
Random Item	5
Random Item	6
Random Item	7
Random Item	8
Random Item	9
Random Item	10
Random Item	11
Random Item	12
Random Item	13
Random Item	14
Random Item	15
Random Item	16

Abbildung 51 ListView mit dynamischen Einträgen und eigenem Layout

1. Platzieren Sie in dem Activity/Fragment wo die ListView angezeigt werden soll ein **ListView-Widget** und geben Sie diesem eine ID (um später die ListView mit Daten abzufüllen).
2. Erstellen Sie nun die Layout-Datei für das Look & Feel eines einzelnen Eintrags im Spinner. Schauen Sie sich dazu die Datei „**res/layout/layout_listview_item**“ an. Dieses Layout beinhaltet zwei TextViews für die Darstellung des Titels und der ID (siehe Abbildung oben).
3. Erstellen Sie nun eine eigene Adapter-Klasse für die Darstellung der Einträge. Diese Klasse muss von der Klasse „ **ArrayAdapter**“ erben. In unserem Fall beinhaltet unser Adapter ItemObject-Objekte (Die Model-Klasse für die ItemObject -Objekte muss vorgängig erstellt werden).

Schauen Sie sich dazu die Klassen „**ItemAdapter.java**“ und „**ItemObject**“ an.

Äusserst wichtig ist der **Konstruktor**. Im Konstruktor vom ItemAdapter wird eine Liste von ItemObjects übergeben. Dies sind die Objekte, welche im Spinner angezeigt werden sollen.

Wichtig ist zudem die Methode **getView(...)**. Diese Methode erhält mit dem Parameter „**position**“ die Position des anzuzeigenden Eintrags in der ListView. Die **getView()**-Methode des Adapters wird immer für alle auf dem Bildschirm sichtbaren Elemente der ListView aufgerufen, nicht jedoch für die nicht sichtbaren (die, welche erst beim Scrollen sichtbar werden)

Der ItemAdapter wurde mit einem **ViewHolder** umgesetzt. ViewHolder ermöglichen, dass das Scrollen in einem Spinner nicht ruckelt. Weitere Infos zu ViewHolder, siehe <https://developer.android.com/training/improving-layouts/smooth-scrolling.html>

4. Zu guter Letzt muss in der **onCreate**-Methode des Activities (**onCreateView** bei Fragment) der Spinner initialisiert und der bei Punkt 3 erstellte Adapter zugewiesen werden (mit **setAdapter(...)**);

Der Code des Beispielprojektes befindet sich auf dem **GitHub-Repository** von Joel Holzer unter dem Namen **ListViewExample**.

8 Daten persistieren

Die meisten Apps speichern (persistieren) Daten auf dem Smartphone. Dies können beispielsweise die Einstellungen einer App sein, oder irgendwelche anderen Daten, welche auf dem Gerät gespeichert und nicht von einem Server geholt werden (z.B. Karten-Daten bei Navigations-Apps, aufgetretene Fehler loggen, etc.)

Häufig ermitteln Apps die Daten, welche diese anzeigen sollen von einem Webservice, d.h. über das Internet. Es ist jedoch nicht sehr performant, wenn eine App die Daten bei jeder Anzeige einer Ansicht neu vom Webservice lädt. Daher holen viele Apps beim ersten Aufruf einer Ansicht die Daten vom Webservice und speichern diese dann lokal auf dem Smartphone ab. Bei allen weiteren Aufrufen werden nur noch die Änderungen der Daten vom Webservice geholt, die übrigen Daten werden lokal geladen. So kann die Datenmenge von Serverabfrage reduziert und somit die Geschwindigkeit einer App massiv erhöht werden.

In diesem Kapitel werden Sie verschiedene Varianten kennenlernen, wie ihre eigene App Daten auf dem Smartphone persistieren kann. Dies ist einerseits die Speicherung von Daten in Dateien, andererseits die Speicherung in einer lokalen Datenbank.

8.1 Daten in Datei persistieren

8.1.1 Shared Preferences

Die einfachste Art, Daten lokal auf dem mobilen Gerät zu speichern, sind die Shared Preferences. Diese ermöglichen die Speicherung von sogenannten Key-Value-Pairs. Das bedeutet, jeder Wert, welcher auf dem Smartphone gespeichert wird, wird unter einem bestimmten, eindeutigen Key in den Shared Preferences abgelegt.

Shared Preferences sind vor allem für die Speicherung von Einstellungen empfehlenswert. Jede Einstellung besteht aus dem Namen der Einstellung und dem Wert dieser Einstellung. Als Key-Value-Pair wären der Name der Key und der Wert der Value.

Der Key ist immer vom Typ „String“. Der Wert (Value) kann vom Typ „String“, „boolean“, „float“ oder „long“ sein.

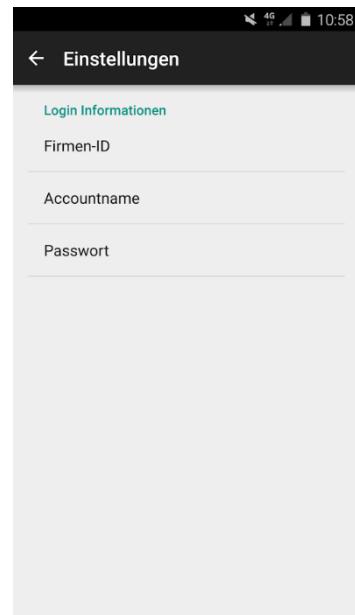


Abbildung 52 Shared Preferences sind optimal für die Speicherung von App-Einstellungen geeignet.

Die Shared Preferences ist eine XML-Datei, welche im App-Data-Ordner der entsprechenden App gespeichert wird. Bei der Erstellung der Shared Preference Datei in der App (siehe Link) kann mit einem Modus angegeben werden, ob die Datei nur von der eigenen App gelesen werden kann oder auch von anderen Apps. Es existieren folgende 3 Modi:

- **MODE_PRIVATE**: Nur die eigene App kann lesen und schreiben
- **MODE_WORLD_READABLE**: Andere Apps können lesen
- **MODE_WORLD_WRITEABLE**: Andere Apps können lesen und schreiben

Mode_WORLD_READABLE und MODE_WORLD_WRITEABLE sind jedoch deprecated (veraltet), weil es gefährlich ist, Daten über Files für andere Apps zur Verfügung zu stellen. Stattdessen sollten für diesen Anwendungsfall Content Provider, Broadcast Receiver und Services erstellt werden.

Unter folgendem Link ist verständlich erläutert wie Daten in den Shared Preferences gespeichert und aus den Shared Preferences gelesen werden können.

<https://developer.android.com/training/basics/data-storage/shared-preferences.html>

8.1.2 Text-Datei in Filesystem speichern

Eine weitere Möglichkeit zur Speicherung von Daten sind Dateien. Dies können Textdateien, Bilddateien, Videodateien oder andere Dateien sein. Android erbt für das Schreiben und Lesen von Dateien die Klassen und Interfaces von Java aus dem Packet „**java.io**.“

Text-Datei in privates Verzeichnis auf Filesystem speichern (interner Speicher)

Um Dateien zu schreiben und zu lesen, in unserem Beispiel eine Textdatei, werden Streams verwendet. In Android geschieht dies über die Klassen **FileOutputStream** und **OutputStreamWriter**. Spezifisch für Android ist zudem die Methode **openFileOutput**. Diese erstellt die gewünschte Datei, sofern diese noch nicht existiert.

Der Methode **openFileOutput** wird wie bei den Shared Preferences ein Modus für die Sicherheitsstufe der Datei übergeben. Zusätzlich zu den 3 Modis der Shared Preferences existiert noch folgender Modus:

- **MODE_APPEND:** Wenn die Datei bereits existiert wird diese um den neuen Inhalt erweitert und nicht wie bei den anderen Modis der Inhalt ersetzt.

Für die Veraltung (deprecated) von **MODE_WORLD_READABLE** und **MODE_WORLD_WRITEABLE** gilt dasselbe wie bei den Shared Preferences.

Nachfolgender Codeausschnitt speichert den String „Hallo Welt“ in das Textfile „helloWorldApp.txt“.

```
private void saveHelloWorldInFile() {
    FileOutputStream fileOutputStream = null;
    OutputStreamWriter outputStreamWriter = null;
    String filename = "helloWorldApp.txt";

    try {
        fileOutputStream = getActivity().openFileOutput(filename, Context.MODE_PRIVATE);
        outputStreamWriter = new OutputStreamWriter(fileOutputStream);
        outputStreamWriter.write("Hallo Welt");

    } catch (FileNotFoundException fileNotFoundException) {
        Log.e("AddNameFragment", "saveHelloWorldInFile()", fileNotFoundException);
    } catch (Exception exception) {
        Log.e("AddNameFragment", "saveHelloWorldInFile()", exception);
    }
    finally {
        if (outputStreamWriter != null) {
            try {
                outputStreamWriter.close();
            }
            catch (IOException ioExceptionOSW) {
                Log.e("AddNameFragment", "outputStreamWriter.close()", ioExceptionOSW);
            }
        }
        if (fileOutputStream != null) {
            try {
                fileOutputStream.close();
            } catch (IOException ioExceptionFOS) {

```

```
        Log.e("AddNameFragment", "fileOutputStream.close()", ioExceptionFOS);
    }
}
}
```

Da kein Speicher-Pfad für das Textfile, sondern nur der Dateiname angegeben wurde, wird das Textfile in das **data**-Verzeichnis der App gespeichert. Für unsere HelloWorld-App ist dies das Verzeichnis:

/data/data/ch.nyp.helloworld/files

Dieses Verzeichnis kann vom Smartphone-User im Datei-Explorer nicht geöffnet werden, es sei denn das Smartphone ist gerootet.

Text-Datei in externem Speicher speichern

Wenn eine Datei auch für andere Apps zugreifbar gemacht werden möchte, sollte sie im externen Speicher abgelegt werden. Der externe Speicher ist entweder eine entfernbare Speicherkarte oder ein bestimmter Bereich im internen Speicher. Eine Datei für andere Apps zugreifbar zu machen kann z.B. bei einem Export aus der App erforderlich sein. Der Export kann dann von einer anderen App oder vom PC wieder importiert werden.

Damit solche Dateien geschrieben werden können, muss die App das Recht **WRITE_EXTERNAL_STORAGE** besitzen. Dazu fügen Sie in der Datei **AndroidManifest.xml** innerhalb des manifest-Blocks folgende Zeile ein.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Nachfolgender Codeausschnitt speichert den String „Hallo Welt“ in das Textfile „helloWorldApp.txt“. Das Textfile wird entweder im Root der SD-Karte, oder falls keine SD-Karte vorhanden ist / unterstützt wird, im externen Speicher-Bereich des internen Speichers (bei meinem Smartphone im Root-Verzeichnis) abgelegt. Mit einer File-Explorer-App können Sie prüfen, ob die Datei auf Ihrem Smartphone gespeichert wurde.

```
private void saveHelloWorldInExternalFile() {  
    String content = "Hallo Welt";  
  
    File file;  
    FileOutputStream outputStream;  
    try {  
        file = new File(Environment.getExternalStorageDirectory(), "helloWorldApp.txt");  
        outputStream = new FileOutputStream(file);  
        outputStream.write(content.getBytes());  
        outputStream.close();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Die wichtigste Methode im obenstehenden Codeausschnitt ist **Environment.getExternalStorageDirectory()**. Die gibt das externe Speicher-Verzeichnis zurück.

Im Gegensatz zum internen Speicher muss der externe Speicher nicht permanent verfügbar sein. Es kann z.B. sein dass der Smartphone-Benutzer die SD-Karte entfernt hat und den USB-Massenspeichermodus aktiviert hat. Ist der USB-Massenspeichermodus auf einem Gerät

aktiviert, so steht der interne Speicher als ein normales Laufwerk zur Verfügung und der externe Bereich des internen Speichers ist nicht verfügbar.

Daher sollte vor jedem Lese- und Schreibzugriff auf den externen Speicher geprüft werden, ob dieser verfügbar ist. Die Verfügbarkeit des externen Speichers kann mit der Methode **Environment.getExternalStorageState()** geprüft werden. Nachfolgender Codeausschnitt zeigt die Überprüfung der Verfügbarkeit vom externen Speicher:

```
String state = Environment.getExternalStorageState();
boolean canRead;
boolean canWrite;
if (Environment.MEDIA_MOUNTED.equals(state)) {
    // lesen und schreiben möglich
    canRead = true;
    canWrite = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // lesen möglich, schreiben nicht möglich
    canRead = true;
    canWrite = false;
} else {
    // lesen und schreiben nicht möglich
    canRead = false;
    canWrite = false;
}
```

8.1.3 Text-Datei aus Filesystem lesen

Auch für das Lesen einer Datei können Sie die Klassen und Interfaces des Pakets „**java.io**.“ verwenden.

Text-Datei aus privatem Verzeichnis (internen Speicher) lesen

Um die im vorhergehenden Kapitel erstellte Textdatei der HelloWorld-App, welche im internen Speicher (privates Verzeichnis der App) erstellt wurde, wieder zu lesen, können Sie die Klassen **FileInputStream**, **InputStreamReader**, **BufferedReader** und die Methode **openFileInput** der Activity-Klasse, verwenden.

Nachfolgender Codeausschnitt liest die Textdatei „helloWorldApp.txt“ und gibt den gelesenen Text in einem Toast aus.

```
private void loadHelloWorldFromInternalStorage() {
    StringBuilder stringBuilder = new StringBuilder();
    FileInputStream fileInputStream = null;
    InputStreamReader inputStreamReader = null;
    BufferedReader bufferedReader = null;
    try {
        fileInputStream = getActivity().openFileInput("helloWorldApp.txt");
        inputStreamReader = new InputStreamReader(fileInputStream);
        bufferedReader = new BufferedReader(inputStreamReader);

        String line;
        // Datei zeilenweise lesen
        while ((line = bufferedReader.readLine()) != null) {
            // Zeilenumbruch hinzufügen
            if (stringBuilder.length() > 0) {
                stringBuilder.append('\n');
            }
            stringBuilder.append(line);
        }
    } catch (FileNotFoundException fileNotFoundException) {
        Log.e("AddNameFragment", "loadHelloWorldFromInternalStorage()", fileNotFoundException);
    }
}
```

```

} catch (Exception exception) {
    Log.e("AddNameFragment", "loadHelloWorldFromInternalStorage()", exception);
} finally {
    if (bufferedReader != null) {
        try {
            bufferedReader.close();
        } catch (IOException ioExceptionBFR) {
            Log.e("AddNameFragment", "bufferedReader.close()", ioExceptionBFR);
        }
    }
    if (inputStreamReader != null) {
        try {
            inputStreamReader.close();
        } catch (IOException ioExceptionISR) {
            Log.e("AddNameFragment", "inputStreamReader.close()", ioExceptionISR);
        }
    }
    if (fileInputStream != null) {
        try {
            fileInputStream.close();
        } catch (IOException ioExceptionFIS) {
            Log.e("AddNameFragment", "fileInputStream.close()", ioExceptionFIS);
        }
    }
}

Toast toast = Toast.makeText(getApplicationContext(), stringBuilder.toString(),
Toast.LENGTH_LONG);
toast.show();
}

```

Text-Datei aus externem Speicher lesen

Um eine Datei vom externen Speicher zu lesen, müssen Sie in der Manifest-Datei Ihrer App die Berechtigung **READ_EXTERNAL_STORAGE** hinzufügen.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Das Lesen funktioniert ähnlich wie beim Lesen der Textdatei aus dem internen Speicher (siehe Code oben). Einzige Änderung ist die Zeile

```
fileInputStream = getActivity().openFileInput("helloWorldApp.txt");
```

welche durch folgende beiden Zeilen ersetzt werden muss:

```
File externalTxtFile = new File(Environment.getExternalStorageDirectory(),
"helloWorldApp.txt");
fileInputStream = new FileInputStream(externalTxtFile);
```

8.1.4 Log-File erstellen und Meldungen loggen mit java.util.Logging

Java.util.Logging kennen Sie vielleicht schon aus Java. Dabei handelt es sich um Klassen zum Loggen von irgendwelchen Meldungen in einer Java-Anwendung, seine es Infomeldungen, Fehlermeldungen, Exceptions, Debugmeldungen, etc.

Jede Anwendung sollte gewisse Infomeldungen, gewisse Fehlermeldungen und zwingend jede Exception loggen. Log-Dateien sind ein hilfreiches Instrument für den Support. Vor allem dienen diese zur Analyse von aufgetretenen Fehlern (Exceptions).

Bei Android gibt es bereits über den App-Store die Möglichkeit, dass der App-User die Entwickler über unerwartete Fehler (Exceptions) informieren kann. Sobald eine App abstürzt, erscheint ein Dialog, wo der App-User den Stack-Trace der aufgetretenen Exception senden kann. Dies ist jedoch freiwillig.

Da über den App-Store nur Exceptions geloggt werden können und dies zudem freiwillig ist, sollte jede App ein zusätzliches Log aufweisen. Dieses soll nicht nur Exceptions loggen, sondern auch gewisse wichtige Schritte in der App als Infomeldungen. Dies vereinfacht die Analyse für Supporter. Das Log-File soll auf dem externen Speicher des Smartphones abgelegt werden, damit über den Dateimanager des Smartphones oder über den Computer darauf zugegriffen werden kann.

Als Beispiel für ein Logging mit java.util.Logging können Sie sich folgende Klassen auf dem Package „logger“ der **LoggingExample-App** (siehe GitHub von Joel Holzer) anschauen:

- **AppLogger:** Singleton-Klasse, welche Meldungen in ein Logfile auf dem externen Speicher speichert.
- **LoggingFormatter:** Legt das Format einer Zeile (Meldung) im Logfile fest.

Damit das Logfile verständlich und nachvollziehbar ist, sollte jede Zeile im Logfile zwingend über folgende Angaben verfügen:

- **Datum und Zeit** wo diese Zeile geloggt wurde (d.h. der Fehler oder die Aktion aufgetreten ist).
- Das **Log-Level** der Meldung, d.h. ob es sich um eine Infomeldung, um eine Debugmeldung, um eine Fehlermeldung (Exception), etc. handelt.
- Der **Name des Loggers**, d.h. die Klasse und je nachdem die Methode, welche die Logmeldung erstellt hat.
- Die **Nachricht**, d.h. eine verständliche Beschreibung der Aktion, der Exception, etc.

Folgender Code loggt in der Demo-App eine Infomeldung:

```
AppLogger.getInstance().logMessage(Level.INFO, "MainActivity.onCreate", "Diese " +
    "INFO wird geloggt");
```

Folgender Code loggt in der Demo -App eine Warning:

```
AppLogger.getInstance().logMessage(Level.WARNING, "MainActivity.onCreate", "Diese " +
    "WARNING wird geloggt ");
```

Folgender Code loggt in der Demo-App eine Exception

```
AppLogger.getInstance().logException("ImageHelper.prepareTypedFileFromBitmap",
    "FileOutputStream zum Speichern eines Beobachtungsfotos auf dem Gerät" +
    " konnte nicht geschlossen werden", e);
```

Wichtig ist, dass im **Manifest**-File folgende Berechtigung eingetragen ist:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Zudem benutzt der AppLogger in der Demo-App die Libraries „Parceler“ und „Apache Commons“. Diese muss im **build.grade** der App im **dependencies**-Block hinzugefügt werden.

```
implementation 'org.parceler:parceler-api:1.1.6'  
implementation 'org.apache.commons:commons-lang3:3.5'
```

8.2 Daten in Datenbank persistieren

Die Möglichkeiten zur Speicherung von Daten in Dateien sind limitiert. Gerade wenn es darum geht, Daten von Objekten, ja vielleicht sogar mehreren tausend Objekten in strukturierter Form und unter Einbezug deren Beziehungen, abzulegen, sind Dateien nicht mehr zu empfehlen. Dazu sind Datenbanken empfehlenswert.

Aus der Praxis kennen Sie sicherlich bereits MySQL und wissen, wie Sie aus einer Java-Anwendung Daten in eine MySQL-DB speichern und auslesen können.

Bei Android gibt es die Möglichkeit, der eigenen App eine SQLite-Datenbank hinzufügen. Dies ist eine dateibasierte Datenbank, welche auf dem Smartphone gespeichert wird. Diese ist nur zugänglich von der eigenen App auf dem Smartphone. Um Daten zu speichern, welche für mehrere Smartphones zugänglich sein sollen, müssen Webservices verwendet werden. Über einen Webservice können Daten in einer zentralen Datenbank im Internet abgelegt werden.

Die meisten Apps verwenden SQLite-Datenbanken für das Zwischenspeichern von grösseren Datenmengen. Initial ermittelt die App die Daten bei einem Webservice, speichert die erhaltenen Daten dann jedoch zur Verwendung in der App in der SQLite-Datenbank ab. So können die Daten innerhalb der App aus der lokalen Datenbank gelesen werden und es ist nicht bei jedem Wechsel oder Neuladen einer Ansicht eine Verbindung zum Webservice notwendig. So kann die Datenmenge von Serverabfragen reduziert und somit die Geschwindigkeit einer App massiv erhöht werden.

8.2.1 Was ist SQLite?

SQLite ist eine dateibasierte, relationale Datenbank. Dateibasiert bedeutet, die Datenbank ist in einer Datei gespeichert und benötigt keinen Datenbankserver wie beispielsweise MySQL oder MSSQL. Eine SQLite-Datenbank ist sehr kompakt und meistens nur ein paar 100 KB gross.

Google hat im Android-Framework viele Klassen zur vereinfachten Erstellung und Anbindung einer SQLite-Datenbank in einer App erstellt. Die SQLite-Datenbank wird immer im privaten Verzeichnis der App gespeichert und ist für andere Apps und auch für den Smartphone-User nicht zugänglich.

8.2.2 Persistence Library

Objektrelationales Mapping ist eine Technik aus der Softwareentwicklung, um Objekte aus einer objektorientierten Anwendung in einer relationalen Datenbank abzulegen.

Vereinfacht gesagt: Java-Objekte in einer relationalen Datenbank speichern ohne SQL-Queries zu schreiben. Um Objektrelationales Mapping einzusetzen, kommen Persistence Libraries zum Zug.

Die meisten Persistence Libraries bilden Klassen auf Tabellen ab. D.h. für jede Java-Klasse, wessen Objekte in der Datenbank gespeichert werden sollen, existiert eine Tabelle in der Datenbank. Die Java-Klassen, welche in der DB gespeichert werden sollen, müssen speziell gekennzeichnet werden. Zwischentabellen existieren meistens nicht als Java-Klassen, sondern können direkt aus der n:m-Beziehung zweier Klassen generiert werden.

Die wohl bekannteste Persistence Libraries für Java sind JPA und Hibernate. Für Android-Apps kommt häufig ORMLite oder **Room** zum Einsatz. Beide Libraries sind kostenlos verfügbar.

Seit kurzem ist Room die offizielle Library des Android Frameworks für den Zugriff auf SQLite-Datenbanken und im SQL-Framework integriert.

O/R Mapping

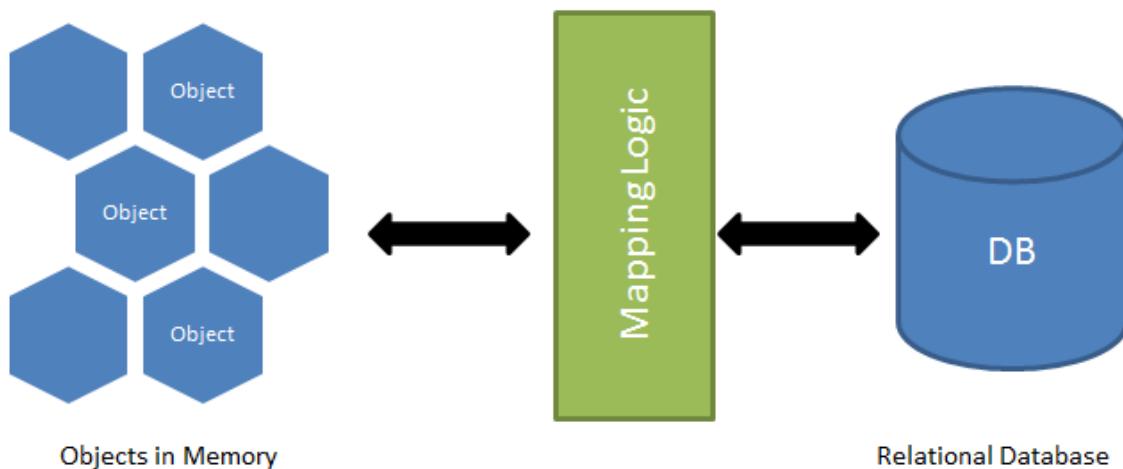


Abbildung 53 Objektrelationales Mapping [16]

8.2.3 Daten mit Hilfe einer Persistence Library in der SQLite-DB speichern

Nachfolgend sehen Sie, wie Sie mit Hilfe von **Room** Daten aus Ihrer App in eine SQLite-Datenbank schreiben können.

Der Code des folgenden Beispielprojektes befindet sich auf dem **GitHub-Repository** von Joel Holzer unter dem Namen **DatabaseExample**.

Schritt 1: Room zum Android-Projekt hinzufügen

Damit Sie Room in Ihrem Projekt verwenden können, müssen sie die benötigten Libraries zu Ihrem Android-Projekt hinzufügen. Fügen Sie dazu im Gradle-File **build.gradle** des Moduls **app** innerhalb von **dependencies** die folgenden Zeilen hinzu:

```
implementation 'android.arch.persistence.room:runtime:1.0.0'
annotationProcessor 'android.arch.persistence.room:compiler:1.0.0'
```

Schritt 2: Model-Klassen erstellen

Erstellen Sie die Java-Klassen (Models), wessen Objekte in der Datenbank gespeichert werden sollen. Meistens haben Sie diese Klassen bereits früher erstellt, da Sie diese ja bereits in Ihrer App benötigen.

Schritt 3: Model-Klassen als Datenbanktabellen markieren

Damit für eine Java-Klasse die entsprechende Datenbanktabelle erstellt wird, muss diese mit verschiedenen **Annotations** als Datenbanktabelle gekennzeichnet werden. Nachfolgend sehen Sie als Beispiel einen Codeauszug aus der Klasse „User“ der App **DatabaseExample**.

```
@Entity
public class User {

    //Autoincrement
    @PrimaryKey(autoGenerate = true)
    public int id;

    @ColumnInfo
    public String accountName;

    @ColumnInfo
    public String firstName;

    @ColumnInfo
    public String middleName;

    @ColumnInfo
    public String lastName;

    @ColumnInfo
    public String street;

    @ColumnInfo
    public String email;

    public int getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```

@Entity markiert die Klasse als Datenbanktabelle. Wird nichts anderes angegeben, so heisst die Tabelle in der DB gleich wie das Model, d.h. in diesem Fall „user“.

@PrimaryKey(autoGenerate = true) markiert die Instanzvariable als PrimaryKey mit Autoincrement.

@ColumnInfo markiert eine Instanzvariable als normale Spalte in der Tabelle. Die Spalte heisst gleich wie die Instanzvariable, wenn nichts anderes angeben wurde.

Es ist auch möglich, die Tabelle oder die Spalten in der DB anders zu benennen als im Model, Attribute aus dem Model nicht in die DB abzubilden, zusammengesetzte Primary Keys zu verwenden, Fremdschlüssel-Beziehungen zu anderen Tabellen abzubilden, und vieles mehr. All dies ist unter folgendem Link erläutert:

<https://developer.android.com/training/data-storage/room/defining-data.html>

Schritt 4: Database-Klasse zum Anlegen/Aktualisieren der Datenbank erstellen

Damit für eine App eine Datenbank erstellt wird, muss die App diese erstellen. Empfehlenswert ist, dass die App beim ersten App-Start die Datenbank erstellt und dann bei jedem weiteren App-Start die Datenbank aktualisiert, falls diese aktualisiert werden muss (z.B. wenn durch das Update der App neue Tabellen dazu gekommen sind).

Erstellen Sie dazu eine Klasse, welche von **RoomDatabase** erbt. Diese Klasse übernimmt dann die Erstellung und Aktualisierung der Datenbank. Als Referenz für die Erstellung dieser Klasse können Sie die Klasse „**AppDatabase**“ aus dem Paket „**persistence**“ der App **DatabaseExample** betrachten.

Diese Klasse soll als Singleton-Klasse erstellt werden, damit nur eine Instanz existieren kann.

```

@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {

    private static final String DB_NAME = "db_demo_db";
    private static AppDatabase appDb;

    public static AppDatabase getAppDb(Context context) {
        if (appDb == null) {
            appDb = Room.databaseBuilder(context, AppDatabase.class, DB_NAME)
                .fallbackToDestructiveMigration()
                .allowMainThreadQueries()
                .build();
        }
        return appDb;
    }

    public abstract UserDao getUserDao();
}
  
```

Schritt 5: SQL-Queries erstellen

Damit aus der Datenbank gelesen und in die Datenbank geschrieben werden kann sind sogenannte **DAO**-Klassen nötig. **DAO** steht für **Data Access Object**.

Als Beispiel können Sie sich das Interface **UserDao** der App **DatabaseExample** anschauen (im Paket **persistence**).

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("DELETE FROM user")
    void deleteAll();

    @Insert
    void insertAll(List<User> users);
}
```

Die Queries werden mit sogenannten Annotationen einer Java-Methode zugeordnet. Wenn das Query ausgeführt werden soll, muss dann einfach die jeweilige Methode aufgerufen werden.

z.B: Aufruf von **getAll()** macht ein Select * auf die User-Tabelle und gibt dann alle ermittelten Daten in Form einer Liste von User-Objekten zurück.

Unter nachfolgendem Link finden Sie weitere Informationen zur Erstellung von Queries und diverse Beispiele (z.B. komplexere Abfragen mit WHERE, etc.)

<https://developer.android.com/training/data-storage/room/accessing-data.html>

Schritt 6: SQL-Queries ausführen

Wenn das Query ausgeführt werden soll, muss die jeweilige Methode des Interfaces aufgerufen werden. Als Beispiel betrachten wir einen Auszug aus der Klasse **MainActivity** in der App **DatabaseExample**.

1. DAO-Instanz erzeugen

Zuerst muss eine Instanz der DAO-Klasse erstellt werden, von welcher Queries ausgeführt werden möchten (d.h. mit Singleton wird aktive Instanz geholt).

```
UserDao mUserDao = AppDatabase.getDb(getApplicationContext()).getUserDao();
```

2. Methode von DAO aufrufen um Query auszuführen

Nachfolgender Code erzeugt zuerst 10 User, fügt diese einer List hinzu und speichert dann die Liste in der Datenbank

```
//10 User-Objekte erstellen und zu List hinzufügen
List<User> usersToSave = new ArrayList<>();
for (int i = 0; i < 10; i++) {
    User user = new User();
    user.firstName = "Vorname" + i;
    user.lastName = "Nachname" + i;
    usersToSave.add(user);
}

//User in DB speichern
mUserDao.insertAll(usersToSave);
```

9 Zugriff auf Webservices

9.1 Was ist ein Webservice?

Im Zeitalter vom Internet, Smartphones und anderen mobilen Geräten (Wearables, etc.) kommt es selten vor, dass ein Computersystem oder eine Software in sich geschlossen ist, d.h. mit keinen anderen Systemen oder Geräten kommuniziert. Häufig besteht ein Gesamtsystem aus vielen Teilsystemen. Jedes Teilsystem führt eine bestimmte Aufgabe aus und kommuniziert mit den anderen Systemen über das Internet, d.h. tauscht mit diesen Daten aus. Ein Teilsystem kann ein Gerät oder eine Softwarekomponente sein.

Als Beispiel nehmen wir eine Wetterstation, wessen Messwerte über eine Webseite und über eine App eingesehen werden können.



Abbildung 54 Beispiel eines verteilten Systems mit Webservices (Wetterstation) [17]

Die Wetterstation misst periodisch die Wetterdaten und schickt diese Daten über das mobile Datennetz an einen Server. Damit die Wetterstation die Daten dem Datenserver geben kann, muss der Datenserver eine Schnittstelle haben. Das ist nun so ein Webservice. Nun wird auf einem anderen Server ein Webserver mit Webseite betrieben. Die Webseite soll die Wetterdaten, die sich nun auf dem Datenserver befinden, anzeigen. Also muss nun der Webserver die Wetterdaten über den Webservice beim Datenbankserver ermitteln. Auch die Smartphone-App, welche die Wetterdaten anzeigen soll, ermittelt die Wetterdaten nun über den Webservice beim Datenbankserver. Der Webservice ist in diesem Fall also die Schnittstelle vom Server gegen aussen.

Durch einen Webservice wird der Zugriff auf den Datenserver im obigen Beispiel vereinheitlicht. Es spielt keine Rolle ob eine Smartphone-App, eine Webseite, eine Java-Anwendung, eine andere Software oder ein anderes System Daten beziehen oder schreiben wollen, alle können denselben Webservice ansprechen. Systeme und Software, welche mit einem Webservice kommunizieren, werden Clients genannt.

Jeder Webservice definiert die Möglichkeiten für die Clients und das Format für die Kommunikation. Die Möglichkeiten für die Clients stellt der Webservice als sogenannte Schnittstellen-Funktionen zur Verfügung. Dies ist vergleichbar mit einer Klasse aus der Programmierung mit öffentlichen, statischen Methoden. Jede Methode hat Parameter, führt eine

Aktion aus und gibt am Schluss etwas zurück. Beispielsweise werden einer Methode zwei Strings übergeben, diese werden zusammengesetzt und dann als zusammengesetzter String zurückgegeben.

So ist es auch bei den Schnittstellen-Funktionen. Ein Webservice stellt z.B. die Schnittstellen-Funktion „getNewBeobachtungenByFirmaId“ zur Verfügung. Der Client ruft diese Funktion des Webservices auf und übergibt ihr die Firmen-ID im definierten Format. Die Funktion wird nun vom Webservice abgearbeitet, d.h. der Webservice ermittelt alle Beobachtungen der übergebenen ID (z.B. aus einer DB) und gibt dann die ermittelten Beobachtungen im definierten Format an den Client zurück.

Auch wenn ein Webservice nicht viel mehr macht als Daten in eine Datenbank zu speichern oder aus dieser zu lesen, macht es dennoch Sinn, einen Webservice einzusetzen und nicht die DB öffentlich zugänglich zu machen und die Daten direkt in diese zu schreiben. Denn ein Webservice bietet die Möglichkeit, gegen aussen nur bestimmte Funktionen anzubieten. Auch bieten Webservices Formate für den Datenaustausch an, welche von jeder Technologie gelesen/geschrieben werden können. Eine weitere Stärke von Webservices sind verschiedene Sicherheits-Funktionen, wie verschiedene Authentifizierungsmöglichkeiten zwischen Client und Webservice und die Verschlüsselung von Nachrichten (HTTPS, etc.).

Die Daten zwischen Client und Webservice werden in einem bestimmten Format ausgetauscht. Dieses ist abhängig von der Art des Webservices. Verbreitete Arten von Webservices sind SOAP-Webservices und REST-Webservices. Bei SOAP werden die Daten im XML-Format ausgetauscht. Bei REST ist das Format nicht strikt geregelt, häufig wird jedoch JSON verwendet.

9.2 SOAP-Webservices

SOAP ist die Abkürzung für Simple Object Access Protokoll und ist ein Netzwerkprotokoll zum Austausch von Daten zwischen Systemen. SOAP-Webservices nutzen dieses Netzwerkprotokoll. SOAP nutzt XML zur Repräsentation der Daten. Übertragen werden die Daten mit http/HTTPS und TCP/IP.

SOAP gibt es schon relativ lange, bereits um das Jahr 2000 herum wurde SOAP 1.0 veröffentlicht. Aktuell sind wir bei Version 1.2.

Lange Zeit war SOAP State of the Art für die Kommunikation in verteilten Systemen (Serviceorientierten Architekturen). Auch heute sind SOAP-Services weiterhin sehr weit verbreitet, vor allem bei der Kommunikation zwischen Computer-Anwendungen und Systemen. Für den Datenaustausch von mobilen Geräten untereinander oder von mobilen Geräten mit Computer-Anwendungen und -Systemen wird SOAP nur selten eingesetzt. Dies hängt vor allem damit zusammen, dass die Daten bei SOAP im XML-Format ausgetauscht werden. XML ist ein Format welches mit Tags aufgebaut ist und einer festen Struktur folgt. Dies bringt jedoch für mobile Anwendungen einige Grenzen mit (mehr dazu später).

Nachfolgende Abbildung zeigt den SOAP-Request (XML) zum Abfragen von Beobachtungen beim XelHa-SOAP-Service:

```

<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ind="http://xelha.ch/webservice/xelha-soap-service/index.php#getNewBeobachtungenByFirma">
  <soapenv:Header/>
  <soapenv:Body>
    <ind:getNewBeobachtungenByFirma soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <firma_id xsi:type="xsd:int">12</firma_id>
    </ind:getNewBeobachtungenByFirma>
  </soapenv:Body>
</soapenv:Envelope>

```

Das XML von SOAP-Requests ist immer recht lang und nur ein kleiner Teil des Requests wird dann schlussendlich auch vom SOAP-Service verarbeitet. Vom obenstehenden XML ist z.B. nur die Zeile <firma_id....>12</firma_id> wirklich relevant, d.h. wird verarbeitet. Die anderen Zeilen definieren die Funktion, welche aufgerufen wird, das Schema und sind Elemente, welche jeder SOAP-Request aufweisen muss.

SOAP-Abfragen sind daher sehr schwergewichtig. Bei verteilten Systemen, welche über Kabelgebundene, schnelle Netzwerkverbindungen verbunden sind, spielt es jedoch nicht so eine grosse Rolle, dass bei jedem Request viele irrelevanten Daten hin- und hergeschickt werden. Sobald aber mobile Geräte beteiligt sind, ist die Geschwindigkeit der Netzwerkverbindungen eingeschränkt. Mobile Geräte kommunizieren mit dem Server vorwiegend über das mobile Datennetz (UMTS, etc.). In Grossstädten ist die Geschwindigkeit meistens sehr gut und vergleichbar mit dem kabelgebundenen Netzwerk. Vielerorts auf dem Land ist die Geschwindigkeit jedoch sehr eingeschränkt.

Aus diesem Grund wurde REST ins Leben gerufen. Bei REST werden zwischen Client und REST-Service fast nur die Daten ausgetauscht, welche wirklich benötigt werden und so die Netzwerkverbindung massiv weniger beansprucht als bei SOAP.

9.3 REST-Webservices

REST ist die Abkürzung von Representational State Transfer. REST wurde auch bereits im Jahr 2000 veröffentlicht, konnte sich aber erst mit dem Aufkommen von mobilen, internetfähigen Geräten wie Smartphones, Tablets, etc. durchsetzen.

REST ist im Gegensatz zu SOAP statuslos, d.h. jede Abfrage bei einem REST-Service steht für sich. SOAP hingegen ist statusbezogen, d.h. verschiedene Abfragen können sich aufeinander beziehen und der Client hat einen bestimmten Status beim Server, welcher in der nächsten Abfrage verändert, etc. werden kann.

Der grösste Unterschied gegenüber SOAP ist jedoch die viel kleinere Datenmenge der Nachrichten und die Struktur der Nachrichten. Auch gibt es bei REST keine Funktionen im Stile von SOAP. Stattdessen arbeitet REST mit Ressourcen.

Ein REST-Service stellt verschiedene Ressource bereit. Ressource sind Objekttypen in der Anwendung (Service). Nehmen wir als Beispiel einen REST-Service, welcher Bücher speichern und ermitteln kann. Dieser REST-Service stellt dann die Ressource „Buch“ zur Verfügung. Für jede Ressource bei einem REST-Service können dann die 4 http-Methoden implementiert und aufgerufen werden:

- **GET:** Ermittelt die Ressource, d.h. ermittelt z.B. ein Buch oder mehrere Bücher
- **POST:** Einer Ressource kann etwas hinzugefügt werden, z.B. wird dem Buch ein Autor hinzugefügt.
- **PUT:** Eine neue Ressource dieses Typs kann erstellt oder der Inhalt ersetzt werden. Z.B. ein neues Buch wird hinzugefügt oder ein bestehendes Buch ersetzt.
- **DELETE:** Eine Ressource kann gelöscht werden, z.B. ein bestimmtes Buch wird gelöscht.

Der REST-Service stellt also verschiedene Ressourcen bereit und dann für jede Ressource maximal die 4 fest definierten http-Methoden. Es müssen aber nicht zwingend alle http-Methoden für eine Ressource definiert werden.

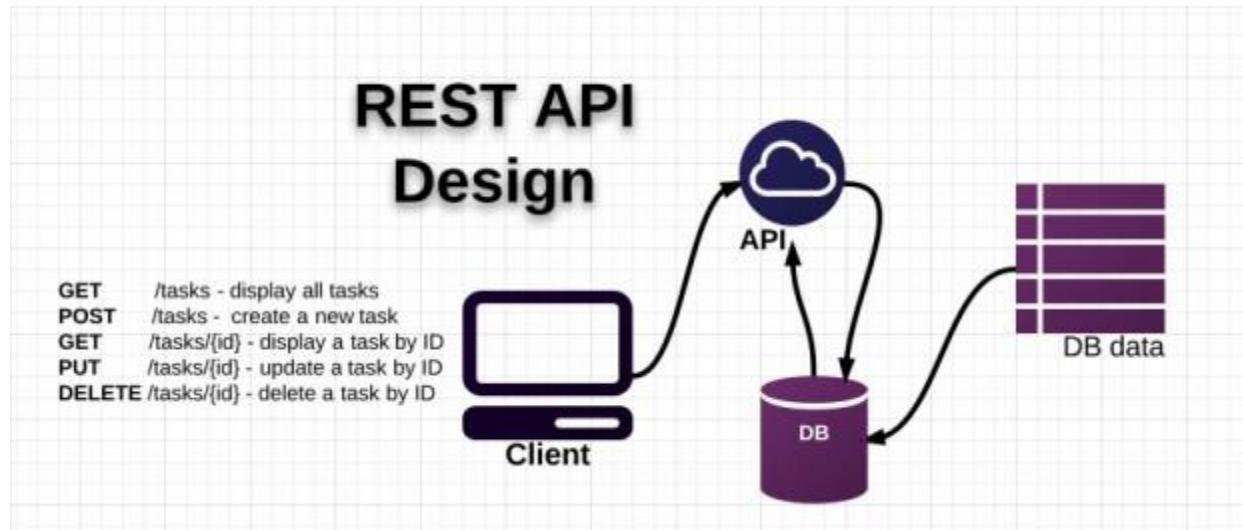


Abbildung 55 REST-Service, oft auch REST API genannt [18]

Die Daten, welche der Client über http an den Webservice sendet (mit POST, PUT, etc.) müssen kein bestimmtes Format aufweisen. Der Webservice muss die Daten einfach verarbeiten können. Ich könnte also für meinen eigenen REST-Service auch ein eigenes Datenformat definieren.

In der Praxis hat sich jedoch **JSON** als Format für den Datenaustausch bei REST-Webservices etabliert.

Nachfolgender Code zeigt, wie der REST-Request für die Abfrage von Beobachtungen einer bestimmten Firma mit JSON aussehen würde:

URL der Ressource: z.B. www.xyz.ch/api/beobachtungen

http-Methode : POST

JSON-Daten: `{"firmen_id":12}`

Im Vergleich zum SOAP-Request werden also für genau das Gleiche statt mehrere 100 Zeichen an Daten nur 16 Zeichen vom Client an den Webservice gesendet.

Weitere Informationen zu JSON finden Sie unter folgendem Link:

<https://wiki.selfhtml.org/wiki/JavaScript/JSON>

9.4 REST-Webservices testen

REST-Webservice können mit dem Google Chrome Plugin „Postman“ getestet werden.

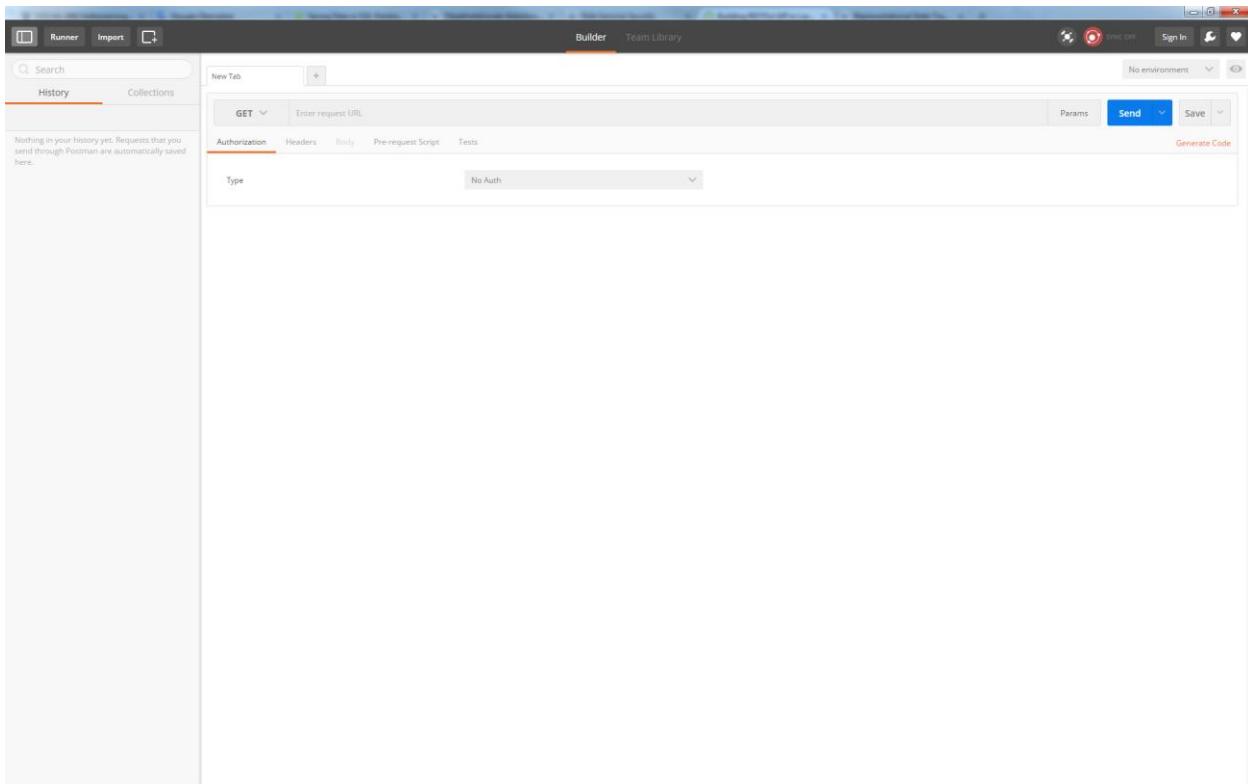


Abbildung 56 Postman Google Chrome Plugin

9.5 Zugriff auf REST-Webservices in Android-App

Nachfolgend wird Ihnen Schrittweise erklärt was Sie in Ihrer App einbauen müssen, damit Sie Daten bei einem REST-Service abfragen und Daten dem REST-Service hinzufügen können.

Für den REST-Webservice-Zugriff in der Android-App empfehle ich die Nutzung der Retrofit-Library. Das Tutorial unter nachfolgendem Link erklärt die Nutzung von Retrofit.

<http://www.vogella.com/tutorials/Retrofit/article.html>

Weitere Informationen zu Retrofit finden Sie unter:

<https://square.github.io/retrofit/>

10 Sensoren

Die meisten heutzutage verbreiteten Android-Geräte verfügen über Sensoren, welche die Bewegung, die Orientierung oder verschiedene Umgebungsbedingungen (Temperatur, Luftdruck, etc.) messen können. Alle diese Sensoren liefern Werte, die Sie in Ihrer eigenen App verarbeiten können.

Das Samsung Galaxy S7 beispielsweise hat einen 3-Achsen-Gyrosensor, einen Beschleunigungssensor, einen Annäherungssensor, einen Umgebungslichtsensor, einen Fingerabdruckscanner, einen Kompass, einen Pulsmesser und einen Barometer.



Abbildung 57 Fingerabdrucksensor bei einem iPhone [19]

In diesem Kapitel lernen Sie, welche Sensoren von Android unterstützt werden und wie Sie diese aus Ihrer App ansprechen und deren Daten verarbeiten können.

10.1 Sensoren bei Android

Die Android-Plattform unterstützt 3 Kategorien von Sensoren:

- **Bewegungssensoren:**
Diese Sensoren messen Beschleunigungs- und Rotationskräfte auf der x-, y- und z-Achse. Zu dieser Kategorie gehören Beschleunigungssensoren, Schwerkraftsensoren, Gyroskop und Drehvektorsensoren.
- **Umgebungssensoren:**
Diese Sensoren messen verschiedene Umweltparameter, wie z.B. die Lufttemperatur, den Luftdruck, die Umgebungsbelichtung oder die Luftfeuchtigkeit. Zu dieser Kategorie gehören Barometer, Photometer, Thermometer, etc.
- **Positionssensoren:**
Messen die Orientierung des Geräts. Zu dieser Kategorie gehören Orientierungssensoren und Magnetometer.

Für den Zugriff auf diese Sensoren stellt Android Klassen im Paket „**android.hardware**“ zur Verfügung.

Die wichtigsten Klassen und Interfaces für den Zugriff auf Sensoren sind:

- **SensorManager:**
Diese Klasse wird benutzt, um eine Instanz eines Sensordiensts zu erstellen. Sie stellt eine Vielzahl von verschiedenen Methoden zur Verfügung um auf Sensoren zuzugreifen, deren Werte auszulesen oder EventListeners für Sensoren zu registrieren. Auch stellt diese Klasse Konstanten zur Verfügung, um die Verfügbarkeit von Sensoren zu prüfen oder Sensoren zu kalibrieren.
- **Sensor:**
Mit dieser Klasse wird eine Instanz für einen spezifischen Sensor erstellt. Diese Klasse beinhaltet Methoden um die Fähigkeiten eines spezifischen Sensors zu ermitteln.

- **SensorEvent:**

Das System benutzt diese Klasse um Sensor-Event-Objekte zu erstellen. Diese Sensor-Event-Objekte stellen Informationen über Ereignisse eines Sensors zur Verfügung.

- **SensorEventListener:**

Sobald der Sensor seinen Wert oder seine Genauigkeit ändert, löst er einen SensorEvent aus. Mit einem SensorEventListener können diese Events abgefangen und entsprechend reagiert werden. Z.B. kann jede Temperaturänderung in einem GUI angezeigt werden.

10.2 Programmierung mit Sensoren in der eigenen App

Der Code des folgenden Beispielprojektes befindet sich auf dem **GitHub-Repository** von Joel Holzer unter dem Namen **SensorDemo**.

10.2.1 Sensoren des Geräts ermitteln

Mit nachfolgendem Code können Sie in Ihrer App alle Sensoren des Geräts ermitteln:

```
SensorManager sensorManager = (SensorManager) getActivity().getSystemService(Context
    .SENSOR_SERVICE);
List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

10.2.2 Luftdruck in einem Toast anzeigen

Um den Wert eines Sensors, in nachfolgendem Beispiel den Luftdruck, auszulesen, muss auf dem **SensorManager**-Objekt für einen bestimmten Sensor einen **SensorEventListener** registriert werden.

Dieser SensorEventListener beinhaltet die Methoden **onSensorChanged** und **onAccuracyChanged**. **onSensorChanged** wird immer dann aufgerufen, wenn der Wert des Sensors ändert.

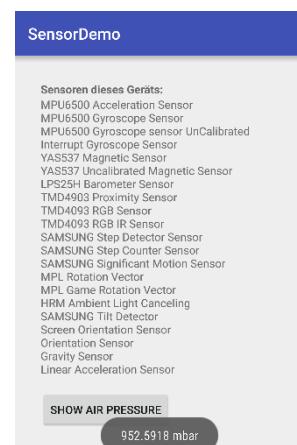


Abbildung 58 Sensor-Demo-App – Luftdruck anzeigen

```
Sensor airPressureSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
if (airPressureSensor != null) {
    mSensorManager.registerListener(mBarometerSensorEventListener, airPressureSensor,
        SensorManager.SENSOR_DELAY_NORMAL);
}
```

Für den Listener, welcher dann den Sensor-Wert des Barometers in einem Toast anzeigt, wird folgender Code benötigt:

```
private SensorEventListener mBarometerSensorEventListener = new SensorEventListener() {
{
    @Override
    public void onSensorChanged(SensorEvent event) {
        float[] sensorValues = event.values;
        String airPressure = sensorValues[0] + " mbar";
        Toast toast = Toast.makeText(getApplicationContext(), airPressure, Toast.LENGTH_LONG);
        toast.show();
        mSensorManager.unregisterListener(this);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
};
```

Die Zeile

```
mSensorManager.unregisterListener(this);
```

macht, dass der Luftdruck nur einmal angezeigt wird. Würde der Listener nicht nach der ersten Ausführung wieder vom SensorManager unregistriert (entfernt) werden, würde bei jeder weiteren Änderung des Luftdrucks die Methode **onSensorChanged** des Listeners erneut aufgerufen.

Mit einem SensorEventListener ist es also möglich, bei jeder Änderung eines Sensorwertes entsprechend zu reagieren. Wie schnell der SensorEventListener auf eine Änderung eines Sensorwertes reagiert ist abhängig von der Auslesefrequenz. Diese Auslesefrequenz wird beim Registrieren eines SensorEventListeners mitgegeben. Für die Auslesefrequenz stehen folgende Konstanten zur Verfügung:

- **SENSOR_DELAY_NORMAL:**
Standard-Auslesefrequenz. Empfehlenswert für Änderungen der Bildschirmorientierung.
- **SENSOR_DELAY_UI:**
Auslesefrequenz empfohlen für das GUI.
- **SENSOR_DELAY_GAME:**
Auslesefrequenz empfohlen für Games.
- **SENSOR_DELAY_FASTEST:**
Reagiert so schnell wie möglich auf eine Änderung des Sensorwertes.

10.2.3 Weitere Sensor-Werte ausgeben

Das Prinzip um Sensoren anzusprechen ist für jeden Sensor dasselbe. Zuerst muss ein SensorManager-Objekt erstellt werden. Auf diesem SensorManager-Objekt muss dann ein SensorEventListener für den gewünschten Sensor registriert werden.

Mit der nachfolgenden Zeile wurde im vorherigen Beispiel der Standardsensor für den Luftdruck ermittelt und für diesen dann später der SensorEventListener registriert.

```
mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
```

Um einen anderen Sensor zu ermitteln, muss einzig der Typ, welcher der Methode `getDefaultSensor` übergeben wird, verändert werden. Welche Sensor-Typen bei Android existieren und welche Konstante Sie aufrufen müssen, um den Sensor für einen bestimmten Wert (z.B. Temperatur) zu holen, finden Sie auf folgender Webseite:

https://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-intro

11 Multitasking

11.1 Threads

Wenn eine App gestartet wird, startet das Android-Betriebssystem einen neuen Prozess mit einem einzelnen Thread für diese App. Standardmäßig laufen also alle Komponenten einer App im gleichen Prozess und im gleichen Thread. Dieser Thread heisst **Main Thread**.

Es gibt jedoch die Möglichkeit, dass eine App in mehreren Prozessen und/oder in mehreren Threads läuft. Es ist beispielsweise möglich, einer App sogenannte Services hinzuzufügen. Ein Service einer App läuft unabhängig davon, ob die App gestartet ist oder nicht in einem eigenen Prozess. Er läuft im Hintergrund und verfügt über kein GUI. Häufig ermitteln Services periodisch Daten bei einem Server. Bei Messenger-Apps beispielsweise (z.B. WhatsApp) läuft in einem eigenen Prozess ein Service, welcher Nachrichten bei einem Server ermittelt. Sobald der Service beim Server eine neue Nachricht ermittelt hat, zeigt er diese auf dem Bildschirm an.

Die meisten Apps laufen jedoch nur in einem Prozess aber in mehreren Threads. Wird eine App gestartet, so läuft immer der Main Thread (UI Thread). Manchmal ist es aber nötig, dass eine App parallel mehrere Aufgaben ausführen muss. Dies kann beispielsweise bei einem Game der Fall sein, wo der App-Benutzer einen Charakter steuert, im Game aber noch weitere Charaktere vorhanden sind, welche sich selbstständig bewegen. Ein weiteres Beispiel sind Abfragen bei einem Server oder komplizierte Berechnungen. Diese können nicht im Main Thread der App ausgeführt werden, da diese sonst das GUI solange „einfrieren“ würden, bis sie abgearbeitet wurden. Als Benutzer möchte ich jedoch während dieser Zeit weiterarbeiten und das GUI weiterbenutzen, weshalb komplizierte Berechnungen und Server-Abfragen immer in einen eigenen Thread ausgelagert werden sollen.

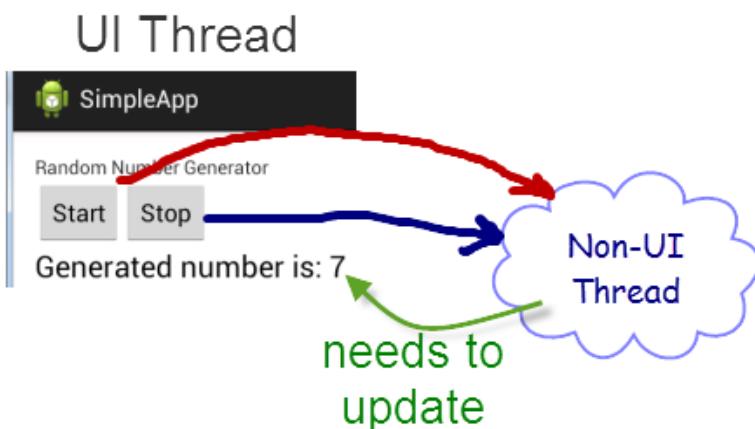


Abbildung 59 Generierung einer Random-Nummer wird in eigenen Thread ausgelagert

11.2 Worker Threads erstellen mit Thread und Runnable

Alle Threads in einer App, bei welchen es sich nicht um den Main Thread (UI Thread) handelt, werden Worker Threads genannt. Da in Android mit Java entwickelt wird und auch die Klassenbibliothek von Java verwendet werden kann, können Threads mit der Klasse **java.lang.Thread** und dem Interface **java.lang.Runnable** erstellt werden.

Nachfolgender Codeausschnitt zeigt, wie bei einem Buttonklick ein Worker Thread erzeugt wird, welcher von einem Server ein Bild herunterlädt und dieses nach Abschluss des Downloads im GUI anzeigt.

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

Mit **start** wird der Thread gestartet. Dieser führt dann die **Run**-Methode aus

Auf den ersten Blick scheint der obenstehende Code korrekt zu sein. Dies wäre er auch, wenn wir nicht im Worker Thread ein GUI-Element des UI Threads ändern würden. Da der Worker Thread aber das geladene Bild einer ImageView hinzufügen möchte, wird die Regel „**do not access the Android UI toolkit from outside the UI thread**“ verletzt.

Um dieses Problem zu umgehen, stellt Android folgende Funktionen bereit, um von einem Worker Thread auf den UI Thread zuzugreifen:

- **Activity.runOnUiThread**
<https://developer.android.com/reference/android/app/Activity.html#runOnUiThread%28java.lang.Runnable%29>
- **View.post**
[https://developer.android.com/reference/android/view/View.html#post\(java.lang.Runnable\)](https://developer.android.com/reference/android/view/View.html#post(java.lang.Runnable))
- **View.postDelayed**
[https://developer.android.com/reference/android/view/View.html#postDelayed\(java.lang.Runnable, long\)](https://developer.android.com/reference/android/view/View.html#postDelayed(java.lang.Runnable, long))

Nachfolgender Codeausschnitt zeigt den korrigierten Worker Thread, wie er korrekt auf den UI Thread zugreift (unter der Verwendung von View.post):

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap =
                loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

11.3 AsyncTasks

Im vorherigen Kapitel haben Sie gesehen, wie Sie unter Verwendung der Klasse `java.lang.Thread` und dem Interface `java.lang.Runnable` ein Worker Thread erstellen können, welcher Daten im GUI anzeigen soll (auf das GUI zugreift).

Bei Android gibt es jedoch noch einen anderen Weg, um in einem Worker-Thread eine Operation auszuführen und dessen Resultat im GUI anzuzeigen und zwar mit einem sogenannten **AsyncTask**. Dies geht ganz ohne Knowhow über die Thread- und Runnable-Klasse von Java und ohne manuelle Threadbehandlung durch den Entwickler.

Um AsyncTasks zu nutzen, müssen Sie eine Klasse oder eine innere Klasse erstellen, welche von der Klasse **AsyncTask** erbt. Die erstellte Klasse/innere Klasse muss folgende Methoden implementieren:

- **doInBackground:**
Führt ihren Codeinhalt in einem Worker Thread aus.
- **onPostExecute:**
Diese Methode erhält das Resultat der Methode `doInBackground` und übergibt dieses an das GUI. Die Methode läuft im UI Thread und kann somit das GUI aktualisieren.

Wird also ein AsyncTask-Objekt erzeugt und dieses dann mit der Methode `execute` ausgeführt, so erstellt die AsyncTask-Klasse automatisch einen neuen Worker-Thread, in welchem der Inhalt der `doInBackground`-Methode ausgeführt wird. Nachdem die `doInBackground`-Methode ausgeführt wurde wird dessen Resultat (return-Wert) automatisch als Parameter an die `onPostExecute`-Methode übergeben, welche dann das GUI im UI-Thread aktualisieren kann.

Nachfolgender Codeausschnitt zeigt, wie bei einem Buttonklick mit einem AsyncTask ein Bild von einem Server geladen und dieses dann in einer ImageView angezeigt wird.

```

public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
     * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
     * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}

```

Ich empfehle Ihnen, dass Sie für Hintergrund-Operationen AsyncTasks anstelle von Threads verwenden.

Weitere Informationen zu AsyncTasks finden Sie auf folgender Webseite:

<https://developer.android.com/reference/android/os/AsyncTask.html>

12 Apps testen

Die Durchführung von ausführlichen Tests ist bei der Entwicklung einer App mindestens so wichtig wie bei der Entwicklung von Software für Computer. Erst wenn eine App alle verschiedenen Arten von Tests erfolgreich durchlaufen hat, ist diese bereit für die Veröffentlichung im App-Store. Eine App mit einer nicht getesteten Funktionalität gehört nicht in den App Store.

Gerade bei Apps ist es wichtig, keine fehlerhaften Apps im App-Store auszuliefern, denn jeder App-Benutzer kann eine App im App-Store bewerten. Apps mit vielen Fehlern weisen relativ schnell ein schlechtes Rating auf, was sich dann auch auf den Download der App auswirkt. Wer lädt schon eine App herunter, wenn diese nur mit 2 von 5 Sternen bewertet ist? Sie sollten daher in jedem App-Projekt genügend Zeit für das Testing einplanen und das Testing auf keinen Fall streichen, wenn die Zeit gegen Ende knapp wird!

In den folgenden Kapiteln werden Sie mehr über verschiedene Testmethoden zum White Box- und Black Box Testing von Android-Apps erfahren.

12.1 White Box Testing von Apps

12.1.1 Was sind White Box Tests?

White Box Tests kennen Sie sicherlich schon aus dem Einsatz bei Java Computer-Applikationen. Bei White Box Tests geht es darum, das System mit Kenntnis über deren innere Funktionsweise zu testen. Das bedeutet, der Tester hat Zugriff auf den Code der App und testet diesen.

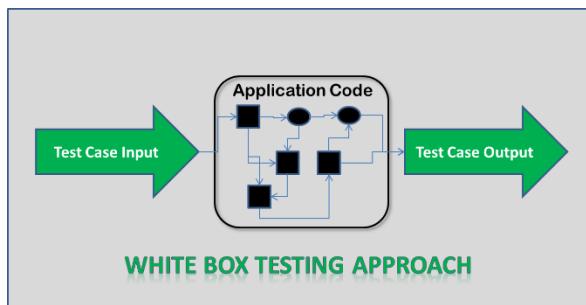


Abbildung 60 White Box Testing [20]

White Box Tests werden genutzt, um einzelne Komponenten/Module einer Applikation zu testen und Fehlern bei diesen aufzudecken. Das Ziel in jeder Applikation ist es, möglichst viele Codezeilen durch White Box Testing zu testen und so schon früh Fehlern bei der Programmierung zu erkennen. White Box Tests werden typischerweise von Software-Entwicklern erstellt, welche den inneren Aufbau der Software kennen.

Der grosse Vorteil von White Box Tests gegenüber Black Box Test ist der Fakt, dass White Box Tests nur einmal geschrieben und dann ohne Aufwand x-Fach ausgeführt werden können. Wurde z.B. in einer Software eine neue Funktion eingeführt, welche möglicherweise nicht erwünschten Einfluss auf das Verhalten einer bestehenden Funktion nimmt, so können einfach die bereits vorhandenen Unit Tests der bestehenden Funktion erneut ausgeführt werden. Konnten diese immer noch erfolgreich durchlaufen werden, so war die Erweiterung korrekt.

Bei jeder Änderung einer Software sollten daher immer alle Unit Tests durchgelaufen werden um zu prüfen ob die bereits bestehende Funktionalität immer noch korrekt funktioniert. Mit Hilfe von Tools kann dieser Schritt automatisiert werden.

Bei Black Box Tests müssten bei einer kleinen Änderung auch alle bereits vorhandenen Funktionen nochmals von Hand durchgetestet werden, damit ein unerwünschtes Fehlverhalten ausgeschlossen werden kann.

In Java werden White Box Tests typischerweise mit dem **JUnit**-Framework erstellt und durchgeführt. Erstellt werden JUnit-Klassen. Diese JUnit-Klassen testen die Funktionsweise der erstellten Klassen und Methoden. Dazu ruft die JUnit-Klasse eine bestimmte Methode mit bestimmten Werten (Parametern) auf und prüft, ob die Methode das gewünschte Resultat zurückliefert. Jede Methode wird dann mit allen möglichen Kombinationen aufgerufen und für jede Kombination geprüft, ob das Verhalten korrekt war. Erhält eine Methode z.B. 2 Strings, setzt diese zusammen und gibt der zusammengesetzte String dann zurück, so ruft die Testmethode die Methode mit verschiedenen Strings auf und prüft ob diese korrekt zusammengesetzt werden.

Am besten können Methoden mit Parametern und Rückgabewerten getestet werden. **Darum sollten Sie wenn möglich ihre Methoden immer so programmieren, dass diese einen Rückgabewert haben.**

12.1.2 Whitebox-Tests bei Android

Android unterscheidet bei Whitebox-Tests zwischen 2 Typen: Unit-Tests und Integration-Tests. Auf die Unit-Tests wird im weiteren Verlauf des Dokuments noch im Detail eingegangen.

Bei den Integration-Tests geht es darum, das Verhalten von GUI-Klassen und Komponenten bei verschiedenen User-Interaktionen zu testen. So kann beispielsweise mit einem mit Java geschriebenen Testfall eine Interaktion eines Users auf ein GUI-Element simuliert und geprüft werden, ob das GUI wie erwartet reagiert. Diese Art von Tests können bei Android mit dem Espresso-Framework (<https://developer.android.com/topic/libraries/testing-support-library/index.html#Espresso>) realisiert werden. Da die Erstellung von codebasierten Integrationstests eher aufwändig und für kleinere Projekte nicht unbedingt empfehlenswert ist, wird diese Art von Test in diesem Dokument nicht mehr weiter behandelt.

12.1.3 Unit-Tests in der eigenen Android-App

Unit Test können hauptsächlich eingesetzt werden, um die Logik einer App zu testen. Getestet wird der Java-Code der App.

Bei Android kann zwischen 2 Arten von Unit-Tests unterschieden werden:

- **Lokale Unit-Tests:**
Diese Unit-Tests werden lokal auf dem Computer ausgeführt. Sie werden von der Java Virtual Machine (JVM) kompiliert und ausgeführt und haben eine schnelle Ausführungszeit. Diese Art von Unit Tests wird für Code verwendet, welcher entweder keine Abhängigkeiten zum Android-Framework aufweist oder bei welchem die Abhängigkeiten durch Mock-Objekte umgangen werden können.
- **Instrumented Unit-Tests:**
Diese Unit-Tests werden auf dem Android-Gerät oder im Emulator ausgeführt. Diese Unit-Tests haben Zugriff auf sogenannte Instrumentation. Instrumentation ist ein Satz von Methoden und Werkzeuge um auf Android-Komponenten zuzugreifen und diese zu steuern. Bei Instrumented Unit-Tests kann beispielsweise auf den Context der App zugegriffen werden. Diese Art von Unit-Tests wird verwendet um Code zu testen, wessen Abhängigkeiten zum Android-Framework nicht mit Mock-Objekten umgangen werden können. Dies können z.B. Methoden von Activities, Fragments, etc. sein.

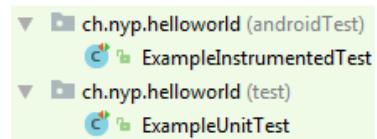
Im weiteren Verlauf dieses Dokuments werden Sie beide Arten von Unit-Tests näher kennenlernen.

Was sind Mock-Objekte? Siehe dazu <https://de.wikipedia.org/wiki/Mock-Objekt>

12.1.4 Lokale Unit-Tests erstellen

Wenn Sie in Android-Studio ein neues Projekt erstellen, erstellt dieses im Java-Verzeichnis automatisch die beiden Verzeichnisse

- androidTest
- test



Im Verzeichnis „test“ werden die lokalen Unit-Tests platziert und im Verzeichnis „androidTest“ die Instrumented Unit-Tests.

Abbildung 61 Unit-Test-Verzeichnisse in App-Projekt

Lokale Unit-Tests werden mit JUnit (Version 4) erstellt.

Paketstruktur der lokalen Unit Tests

Wichtig ist, dass im test-Verzeichnis dieselbe Paketstruktur herrscht wie im main-Verzeichnis (Programmcode). Nachfolgende Tabelle zeigt als Beispiel den Pfad dreier Java-Klassen und deren lokaler Unit-Test-Klassen.

Pfad Java-Klasse	Pfad Unit-Test-Klasse
src/main/java/ch/nyp/junitdemo/helper.java	src/test/java/ch/nyp/junitdemo/helper.java
src/main/java/ch/nyp/junitdemo/book.java	src/test/java/ch/nyp/junitdemo/book.java
src/main/java/ch/nyp/junitdemo/author.java	src/test/java/ch/nyp/junitdemo/author.java

Abbildung 62 Lokale Unit-Tests Paketstruktur

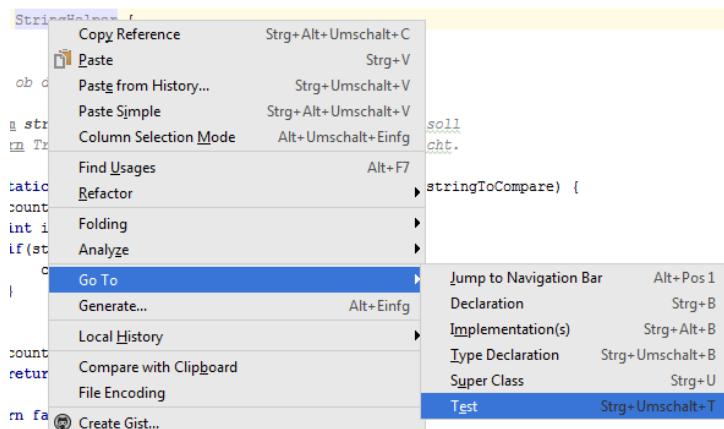
Abhängigkeiten (Libraries) hinzufügen

Um lokale Unit-Tests erstellen und ausführen zu können, müssen Sie in dem **build.gradle** das **JUnit-Framework** in der Version 4 importieren. Zudem müssen Sie das **Mockito-Framework** importieren, wenn Sie Unit-Tests mit Mock-Objekten erstellen möchten. Fügen Sie dazu die folgenden Codezeilen innerhalb des dependencies-Block im build.gradle ihrer App hinzu:

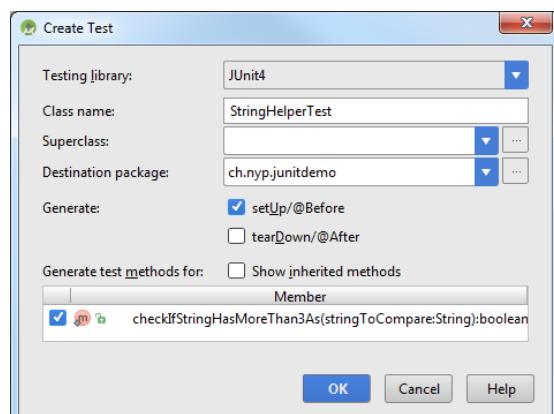
```
testImplementation 'junit:junit:4.12'
// Optional -- Mockito framework
testImplementation 'org.mockito:mockito-core:1.10.19'
```

Unit Test Klasse erstellen ohne Mock-Objekte

Ich empfehle Ihnen für jede zu testende Java-Klasse eine JUnit-Testklasse zu erstellen. Eine Testklasse können Sie erstellen, indem Sie einen **Rechtsklick auf den Klassennamen** der Java-Klasse, dann im Kontextmenü „**Go To → Test**“ auswählen.



Danach wählen Sie im aufgehenden kleinen Fenster „**Create New Test...**“ und wählen nun aus, was alles in der Testklasse bereits generiert werden soll. Nachdem Sie **OK** geklickt haben, müssen Sie noch angeben, dass Sie die generierte JUnit-Testklasse im **test-Verzeichnis** ablegen wollen und schon erstellt Android-Studio automatisch die Testklasse.



Nachfolgende sehen Sie den JUnit-Testcode zum Testen der Methode `StringHelper.checkIfStringHasMoreThan3As`. Diese Methode gibt True zurück wenn der übergebene String mehr als 3 A beinhaltet, sonst false. Darum rufen wir im JUnit-Test diese Methode mit verschiedenen Strings auf (Strings mit mehr und weniger A) und prüfen ob die Methode den richtigen Return-Wert liefert.

```

@Test
public void testCheckIfStringHasMoreThan3As() throws Exception {
    String stringWith2As = "bbAbbcccAwleiie";
    String stringWith3As = "bbAbbcccccAcdA";
    String stringWith4As = "bbbbbbAccdeeeeeAA";
    String stringWith3AsInARow = "AAAAbbbdkdkdkd";
    String stringWith4AsInARow = "AAAAAbbbdkdkdkd";

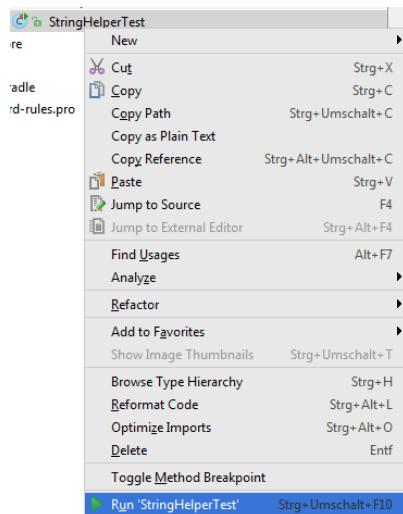
    assertFalse(StringHelper.checkIfStringHasMoreThan3As(stringWith2As));
    assertFalse(StringHelper.checkIfStringHasMoreThan3As(stringWith3As));
    assertTrue(StringHelper.checkIfStringHasMoreThan3As(stringWith4As));
    assertFalse(StringHelper.checkIfStringHasMoreThan3As(stringWith3AsInARow));
    assertTrue(StringHelper.checkIfStringHasMoreThan3As(stringWith4AsInARow));
}
  
```

Modul 335: Mobile-Applikationen realisieren

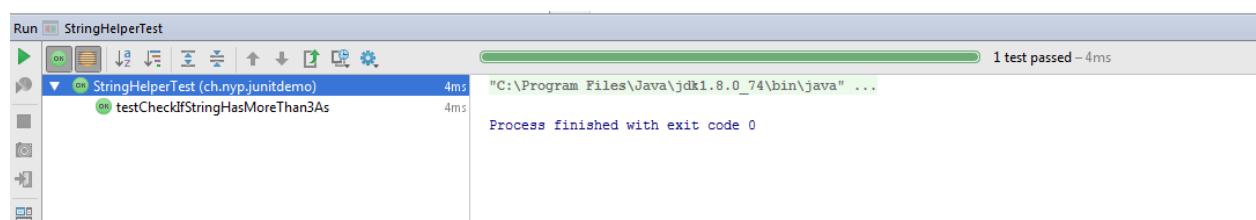
Jede Testmethode muss mit der Annotation `@Test` gekennzeichnet sein. Zudem können Sie mit `assert...` die Überprüfungen machen. Die Asserts entsprechen den JUnit Asserts. Weitere Infos dazu finden Sie auf folgender Webseite:

<http://junit.sourceforge.net/javadoc/org/junit/Assert.html>

Um den JUnit-Test auszuführen, klicken Sie Rechtsklick auf die JUnit-Testklasse und dann „Run...“.



Nun führt Android alle Testmethoden der Testklasse aus. Die Testergebnisse werden unten in Android-Studio in einem Fenster angezeigt.



Unit Test Klasse erstellen mit Mock-Objekte

Auf lokale Unit Tests mit Mock-Objekten wird an dieser Stelle nicht weiter eingegangen. Von Ihnen wird nicht erwartet, dass Sie solche Tests in Ihrem eigenen Projekt umsetzen.

Falls es Sie dennoch interessiert, finden Sie weitere Informationen auf folgender Webseite:

<https://developer.android.com/training/testing/unit-testing/local-unit-tests.html>

12.1.5 Instrumented Unit-Tests erstellen

Wie bereits erwähnt, werden Instrumented Unit-Tests im Verzeichnis „**androidTest**“ abgelegt.

Wie die lokalen Unit Tests werden auch Instrumented Unit-Tests mit JUnit 4 erstellt. Zusätzlich dazu können aber noch weitere Klassen & Methoden benutzt werden, die Android zur Verfügung stellt.

Paketstruktur der Instrumented Unit-Tests

Wichtig ist, dass im androidTest-Verzeichnis dieselbe Paketstruktur herrscht wie im main-Verzeichnis (Programmcode). Nachfolgende Tabelle zeigt als Beispiel den Pfad dreier Java-Klassen und deren lokaler Unit-Test-Klassen.

Pfad Java-Klasse	Pfad Unit-Test-Klasse
src/main/java/ch/nyp/junitdemo/helper.java	src/androidTest/java/ch/nyp/junitdemo/helper.java
src/main/java/ch/nyp/junitdemo/book.java	src/androidTest/java/ch/nyp/junitdemo/book.java
src/main/java/ch/nyp/junitdemo/author.java	src/androidTest/java/ch/nyp/junitdemo/author.java

Abbildung 63 Instrumented Unit-Tests Paketstruktur

Abhängigkeiten (Libraries) hinzufügen

Um Instrumented Unit-Tests erstellen und ausführen zu können, müssen Sie in dem **build.gradle** ein paar Libraries importieren. Fügen Sie dazu die folgende Codezeilen innerhalb des dependencies-Block im build.gradle ihrer App hinzu:

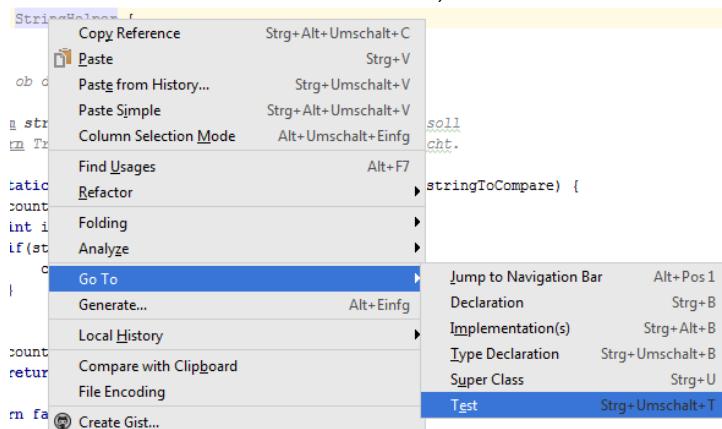
```
androidTestImplementation 'com.android.support.test:runner:1.0.1'
androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'
```

Zudem müssen Sie im defaultConfig-Block im gleichen Gradle-File folgende Zeile hinzufügen:

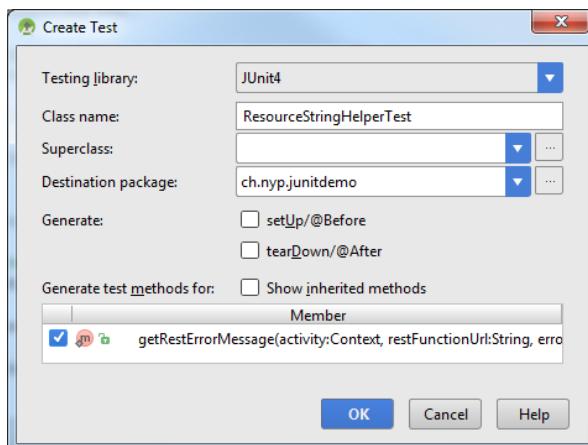
```
testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
```

Instrumented Unit Test Klasse erstellen

Eine Instrumented Unit Test Klasse können Sie erstellen, indem Sie einen **Rechtsklick auf den Klassennamen** der Java-Klasse, dann im Kontextmenü „**Go To → Test**“ auswählen.



Danach wählen Sie im aufgehenden kleinen Fenster „**Create New Test...**“ und wählen nun aus, was alles in der Testklasse bereits generiert werden soll. Nachdem Sie **OK** geklickt haben, müssen Sie noch angeben, dass Sie die generierte JUnit-Testklasse im **androidTest-Verzeichnis** ablegen wollen und schon erstellt Android-Studio automatisch die Testklasse.



Nachfolgend sehen Sie einen Instrumented Unit Test. Dieser erbt von der Klasse `ActivityInstrumentationTestCase2`. Dies ist nötig, weil die Testmethode eine Activity instanziiieren muss, welche dann der Methode, welche getestet wird, übergeben wird.

```
public class ResourceStringHelperTest extends
ActivityInstrumentationTestCase2<MainActivity> {

    public ResourceStringHelperTest() {
        super(MainActivity.class);
    }

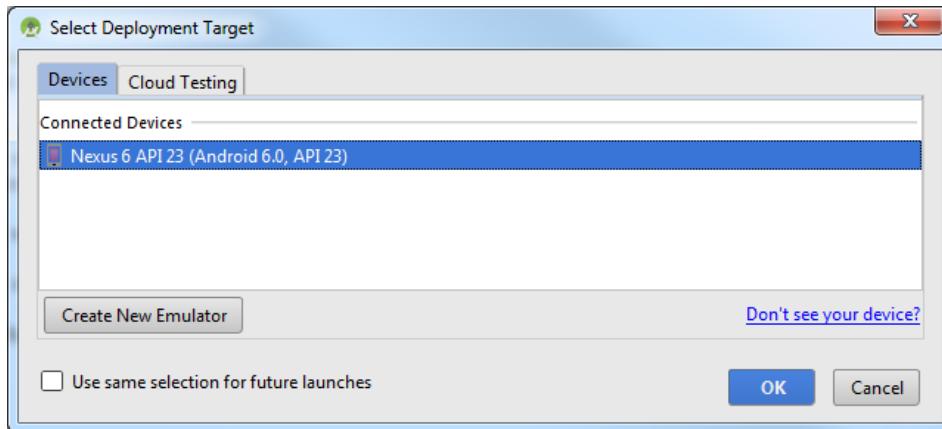
    @UiThreadTest
    public void testGetRestErrorMessage() throws Exception {

        MainActivity mainActivity = getActivity();
        String restUrl = "/lernenden";
        int errorCode = 400;

        String expectedString = "Mindestens eine der gemachten Angaben ist fehlerhaft
oder es wurden mehr als 2 Fotos zum Upload angegeben.";
        String stringFromLanguageFile =
ResourceStringHelper.getRestErrorMessage(mainActivity,
            restUrl, errorCode);
        assertEquals(expectedString, stringFromLanguageFile );
    }
}
```

Getestet wird die Methode `getRestErrorMessage` der Klasse `ResourceStringHelper`. Dieser wird eine URL eines REST-Request und ein Error-Code übergeben. Daraus generiert die Methode dann einen Key für das String-File, liest den Wert dieses Keys aus dem String-File und gibt den String aus dem String File zurück.

Um den Instrumented Test auszuführen, klicken Sie Rechtsklick auf die Testklasse und dann „**Run...**“. Da der Instrumented Test entweder im Emulator oder auf dem angeschlossenen Gerät ausgeführt wird, muss nun noch ausgewählt werden, wo er ausgeführt werden soll.



12.2 Black Box Testing von Apps

12.2.1 Was sind Black Box Tests?

Mit Black Box Tests wird ein System/eine Software getestet, ohne dass der Tester die innere Funktionsweise eines Systems kennt. Der Tester testet die Software funktionsorientiert, d.h. prüft ob die Software die Funktionen korrekt ausführt. Wie die Software aufgebaut ist und welcher Code zur Abarbeitung einer Funktion aufgerufen wird, interessiert den Tester bei Black Box Tests nicht.

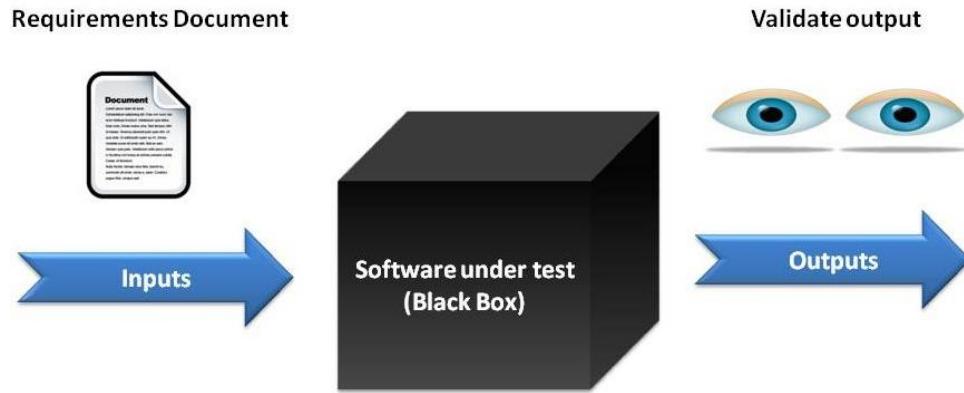


Abbildung 64 Black Box Testing [21]

Das Ziel von Black Box Tests ist es, das System/die Software mit der Spezifikation zu prüfen. Die Spezifikation definiert, welches Ergebnis erwartet wird. Die Spezifikation kann z.B. aus Use Cases (für die Funktionen & verschiedenen Szenarien) und aus GUI-Entwürfen (Look & Feel und Aufbau des GUIs) bestehen.

Es gibt verschiedene Arten von Black Box Tests. In diesem Dokument werden **Funktionstests** und **GUI-Tests** behandelt.

Mit Funktionstests werden alle Szenarien der Use Cases durchgetestet, d.h. es wird geprüft ob die Software für die unterschiedlichen Szenarien wie erwartet reagiert. Wird z.B. ein Kontaktformular getestet, so wird geprüft, ob das Kontaktformular korrekt abgesendet wird und ob bei fehlerhaften Eingaben die korrekten Fehlermeldungen erscheinen.

Bei den GUI-Tests hingegen testen die Tester ob alle GUIs wie in den GUI-Entwürfen definiert umgesetzt wurden. Dabei geht es auch darum, das GUI auf Bildschirmen mit verschiedenen

Auflösungen, Seitenverhältnisse und Orientierung zu testen. Der Fokus der GUI-Tests liegt auf der Korrektheit des GUI-Aufbaus und der Darstellung von Elementen, Bildern und Schriften.

12.2.2 Funktionstests

Die Durchführung der Funktionstest geschieht nach einem strikten Ablauf, nach sogenannten Testfallbeschreibungen. Für jeden Testfall existiert eine Testfallbeschreibung. Diese wird auf der Basis der Use Cases erstellt. **Jedes Szenario der Use Cases muss von mindestens einem Testfall abgedeckt werden.** Die Testfallbeschreibung beinhaltet:

- Nr. des Testfalls
- Nr. des Use Case
- Beschreibung der Testumgebung (welche Version wird getestet, welche Daten sind vorhanden, etc.)
- Beschreibung der zu testenden Funktionalität
- Datum und Tester (Wird vom Tester bei der Durchführung ausgefüllt)
- Testschritte. Jeder Testschritt besteht aus:
 - o Nr.
 - o Aktion (Verhalten des Testers)
 - o Erwartetes Ergebnis
 - o Effektives Ergebnis (Wird vom Tester bei der Durchführung ausgefüllt)
 - o Erfüllt / Nicht Erfüllt (Wird vom Tester bei der Durchführung ausgefüllt)
 - o Kommentar

Beispiel-Testfall für die Login-Ansicht einer App

Testfall-Nr	1				
Testfall-Bezeichnung	Login				
Use Case Nr.	1				
Testumgebung	XelHa-App 1.0 Samsung Galaxy S6 mit Android 6.0.1				
Zu testende Funktionalität	Testet die Login-Ansicht der XelHa-App. Getestet werden alle Szenarien des Use Cases 1 des Dokuments „LINK ZU DOKUMENT“.				
Datum der Testdurchführung	01.06.2016				
Tester	Hans Muster				
Testschritte:					
Nr.	Aktion	Erwartetes Ergebnis	Effektives Ergebnis	Erfüllt	Kommentar
1	Öffnen Sie die XelHa-App	Welcome-Ansicht wird für ca. 1 Sek angezeigt. Danach erscheint die Login-Ansicht.	Wie erwartet	Ja	

2	Geben Sie folgende Logindaten ein: Accountname: krebs Passwort: 123456 Klicken Sie auf den Button „Login“.	Das Login wird durchgeführt. Mit einem Progress-Dialog wird dies dem User mitgeteilt. Nach ein paar Sekunden war das Login erfolgreich und die Startseite der App erscheint.	Es erscheint nicht die Startseite, sondern die Seite zum Hinzufügen einer Beobachtung	Nein	
	Usw.				

Tabelle 4 Beispiel-Testfall

12.2.3 GUI-Tests

Im Kapitel 6.3 haben Sie gesehen, dass für Android eine Vielzahl an Smartphones und Tablets mit verschiedenen Bildschirmgrößen und Auflösungen existieren. Auch kann eine App entweder im Portrait-Modus oder im Landscape-Modus ausgeführt werden.

In der Konzeptphase eines Projekts wird zusammen mit dem Auftraggeber definiert, wie sich die App auf den verschiedenen Bildschirmgrößen, Auflösungen und Bildschirmorientierungen verhalten soll und für welche Geräte die App optimiert werden soll. Um nach der Realisierung der App zu gewährleisten, dass die Anforderungen an das GUI alle erfüllt werden, muss das GUI spezifisch getestet werden.

Die GUI-Tests können entweder auf Android-Geräten oder im Emulator durchgeführt werden. Getestet werden muss die App für alle in den Anforderungen definierten Bildschirmgrößen, Auflösungen und Orientierungen. Da eine Vielzahl von unterschiedlichen Bildschirmgrößen und Auflösungen existieren, können unmöglich alle getestet werden. Dies ist aber auch nicht nötig, da Android die generalisierten Pixeldichten (ldpi, mdpi, hdpi, etc.) eingeführt hat. Daher sollten Sie eine App einfach auf einem Smartphone/Emulator pro Pixeldichte testen.

Nachfolgende Tabelle zeigt jeweils ein Gerät pro Bildschirmklasse (ab hdpi):

Smartphone	Bildschirmklasse	Höhe in px	Breite in px
Samsung Galaxy S6	xxxhdpi	2560	1440
LG Nexus 5	xxhdpi	1794	1080
Samsung Galaxy S3	xhdpi	1280	720
Samsung S3 mini	hdpi	800	480

Tabelle 5 Geräte pro Bildschirmklasse

12.3 Testvorgehen in einem App-Projekt

Sie haben verschiedenen Testverfahren für das Testing von Android-Apps kennengelernt. Um die Qualität einer App zu gewährleisten, sollten Sie grossen Wert auf das Testing legen und dieses strukturiert durchführen. Das ganze Testing beinhaltet nicht nur die Durchführung von Tests. Dazu gehört auch die Planung von Tests (Konzept, Ressourcen planen, etc.), die Beschreibung von Testfällen, die Beschreibung der Testergebnisse und die Dokumentation von Fehlern. In Ihrer Ausbildung zum Informatiker wurden Sie bereits mehrfach in das Testing von Applikationen eingeführt. Dieses Kapitel dient als Erinnerung und zeigt das Vorgehen beim Testing einer Android-App auf. Es wird jedoch nicht auf alle Punkte im Detail eingegangen, da gewisses Wissen vorausgesetzt wird.

12.3.1 Testkonzept

Die erste Phase des Testings ist das Testkonzept. Es bildet die Grundlage für die Vorbereitung und Durchführung der Tests. Im Testkonzept wird folgendes festgelegt:

- **Testziele:**
Welche Ziele sollen durch das Testing erreicht werden?
- **Testobjekte:**
Welche Komponenten oder Funktionen der App werden getestet?
- **Testarten:**
Welche Arten von Tests werden durchgeführt (siehe oben, z.B. welche Arten von White Box und Black Box Tests)? Welche Komponenten/Funktion wird mit welcher Art von Test getestet?
- **Testinfrastruktur:**
Welche Testinfrastruktur wird benötigt? Welche Systeme und Testdaten müssen für welche Tests bereitgestellt werden?
- **Testorganisation:**
Wie ist das Testteam organisiert? Wer führt welche Tests durch? Wer ist für was verantwortlich?
- **Testplanung:**
Welcher Zeitplan ist einzuhalten? Welche Tests werden wann ausgeführt?
- **Testfallbeschreibung:**
Eine detaillierte Beschreibung der Testfälle. Bei einem Testfall geht es immer darum, denn Soll-Zustand mit dem Ist-Zustand zu vergleichen. Es gibt White Box Testfälle und Black Box Testfälle. Jeder Testfall besteht aus mehreren Testschritten.

Nachfolgend werden gewisse der genannten Punkte des Testkonzepts im Hinblick auf das Testing einer App im Detail angeschaut.

Testarten

In den vorhergehenden Kapiteln haben Sie verschiedene Testarten kennengelernt. Bei einer App sollten Sie immer folgende Arten von Tests durchführen:

- **Unit Tests (Lokale und Instrumented):**

Damit soll der Programmcode getestet werden. Sie sollten zumindest die Logik Ihrer App mit Unit Tests testen. Zur Logik gehören Helper-Klassen, aber auch Methoden in Activity- oder Fragment-Klassen, welche Operationen oder Berechnungen durchführen. Alle diese Tests sollten, wenn die App überarbeitet wird, erneut durchgeführt werden. Die Unit Tests sind während der Entwicklung zu erstellen und durchzuführen. Sobald die Entwicklung abgeschlossen wurde, sollten nochmals alle Unit Tests durchgeführt werden.

- **Funktionstests:**

Die Funktionstests basieren auf den Use Cases. Getestet wird damit die Funktionalität einer App. Mit den Funktionstests sollten Sie Ihre App für alle Szenarien ihrer Use Cases (Hauptszenarien, Alternativszenarien, Fehlerszenarien) durchklicken. Die Funktionstests führen Sie entweder auf Smartphones oder im Emulator aus. Wichtig ist, dass Sie die Funktionstests auf verschiedenen Betriebssystemversionen durchführen. Alle Betriebssystemversionen zu testen, welche Ihre App unterstützen soll, ist aus Zeitgründen meistens nicht möglich. Testen Sie daher die Betriebssystemversionen, wessen Änderungen aufgrund der benutzten Funktionen in der App einen Einfluss auf ihre App haben könnten.

Testen Sie nicht nur Ihre App mit Funktionstests, sondern auch andere Systemkomponenten, welche Sie entwickelt haben, beispielsweise ein entwickelter REST-Webservice.

- **GUI-Tests:**

Wie Sie bereits wissen, existieren für Android eine Vielzahl an verschiedenen Geräten mit unterschiedlichen Bildschirmgrößen. Zudem können alle Geräte im Portrait- und im Landscape-Modus verwendet werden. In der Analyse- oder Designphase einer App wird zusammen mit dem Kunden definiert, für welche Bildschirmgrößen (generalisierte Pixeldichten von Android (hdpi, etc.)) und für welche Orientierungen die App optimiert sein soll. Auch muss definiert werden, ob die App speziell für Tablets optimiert wird (z.B. Darstellung von List-Detail-View auf einer Bildschirmseite, etc.).

Bei den GUI-Tests sollten Sie testen, ob die App nun auf allen definierten Pixeldichten und Orientierungen funktioniert und alle Ansichten (dazu gehören auch Dialoge, Toasts, etc.) wie gewünscht angezeigt werden / optimiert wurden. Klicken Sie dazu das GUI in Emulatoren unterschiedlicher Pixeldichte und Ausrichtung durch.

Testfallbeschreibung

Eine Testfallbeschreibung sollten Sie für die White Box und die Black Box Tests erstellen. Bei der Testfallbeschreibung geht es darum, die Testfälle mit Testschritten zu beschreiben. Eine Testfallbeschreibung besteht immer aus

- Nr. des Testfalls
- Use Case Nr.
- Beschreibung der Testumgebung (welche Version wird getestet, welche Daten sind vorhanden, etc.)
- Beschreibung der zu testenden Funktionalität / Ansichten / Klasse & Methode bei Unit Tests
- Datum und Tester (Wird vom Tester bei der Durchführung ausgefüllt)
- Erfüllt? (Wird vom Tester bei der Durchführung ausgefüllt)
- Erwartetes Ergebnis
- Effektives Ergebnis

12.3.2 Testprotokoll

Wichtig ist, dass sowohl für die Durchführung von White Box und Black Box Test ein Testprotokoll erstellt wird. Das Testprotokoll sind die ausgefüllten Testfallbeschreibungen.

Im Testprotokoll ist auch ersichtlich, welche Teste wann wiederholt wurden und wie sich das Ergebnis der Tests bei den Wiederholungen der Tests nach dem Bugfixing verändert hat.

13 Apps veröffentlichen

In diesem Kapitel lernen Sie, wie Sie eine eigene App im App Store von Google veröffentlichen und somit einem breiten Publikum zugänglich machen können. Zudem lernen Sie gewisse Tools des App-Stores (u.a. Statistik) kennen.

13.1 Google Play Store

13.1.1 Was ist der Google Play Store?

Google Play Store ist der offizielle App-Store von Google zur Veröffentlichung von Apps. Apps im Google Play Store können entweder über den Webbrowser, oder über die Google Play App bezogen/installiert werden. Die Google Play App ist auf jedem Android-Smartphone vorinstalliert.

Es gibt auch noch andere App-Stores für die Verteilung von Android-Apps. Auf diese wird jedoch in diesem Dokument nicht weiter eingegangen.



Abbildung 65 Icon vom Google Play Store

13.1.2 Lizenzierungsmodelle einer App im Google Play Store

Es gibt zwei Möglichkeiten, wie Sie mit einer App im Google Play Store Geld verdienen können:

- **Kostenpflichtiger Download:** Der Download der App ist kostenpflichtig
- **In-App-Käufe:** Der Download der App ist kostenlos, in der App können jedoch In-App-Käufe getätigter werden. In-App-Käufe sind vor allem in Games weit verbreitet, wo der Benutzer beispielsweise Fähigkeiten für seinen Charakter einkaufen oder Levels überspringen kann. Auch wiederkehrende Lizenzen (z.B. jährliche Lizenzen) können über In-App-Käufe abgewickelt werden. Bei der XelHa-App wird beispielsweise dem User pro Jahr 2.50 CHF für die Lizenz berechnet (wiederkehrender In-App-Kauf)

Natürlich ist es auch möglich, dass der Download kostenpflichtig ist und zusätzlich dazu noch mit In-App-Käufen Geld verdient wird.

Von den Einnahmen des Downloads und der In-App-Käufe gehen 70% an den Entwickler der App, 30% behält Google (Transaktionsgebühr).

13.2 Entwicklerkonto erstellen und Store-Eintrag vorbereiten

13.2.1 Entwicklerkonto erstellen

Damit Sie Apps im Google Play Store veröffentlichen können, müssen Sie sich als Publisher in der [Google Play Developer Console](#) registrieren. Diese Registrierung kostet einmalig 25 Dollar. Sie können dann so viele Apps veröffentlichen wie Sie wollen.

13.2.2 Google Payments Merchant Account erstellen

Wenn Sie mit Ihrer App Geld verdienen, sei es durch den kostenpflichtigen Download oder In-App-Käufe, müssen Sie einen Google Payments Merchant Account erstellen.

Eine Anleitung dazu finden Sie auf folgender Webseite:

<https://developer.android.com/distribute/googleplay/start.html>

13.2.3 Store-Eintrag vorbereiten

Sobald Sie das Entwicklerkonto erstellt haben, können Sie sich mit dem erstellten Konto in der [Google Play Developer Console](#) einloggen.

Nach dem Login sehen Sie unter „Alle Apps“ Ihre eigenen Apps. Über den Button „**„Neue App hinzufügen“**“ können Sie unter Angabe des Titels der App einen Store-Eintrag für Ihre App vorbereiten.

Beim Store-Eintrag müssen Sie nun eine Kurzbeschreibung, eine vollständige Beschreibung und Screenshots ihrer App hochladen. Zudem müssen Sie Symbole und Banner für den Google-Play Store hochladen und die App kategorisieren (App-Typ, Kategorie, Einstufung des Inhalts) und Kontaktinformationen und eine Datenschutzerklärung hinterlegen. Alle diese Angaben sind Muss-Felder und müssen ausgefüllt werden, bevor die App hochgeladen werden kann.

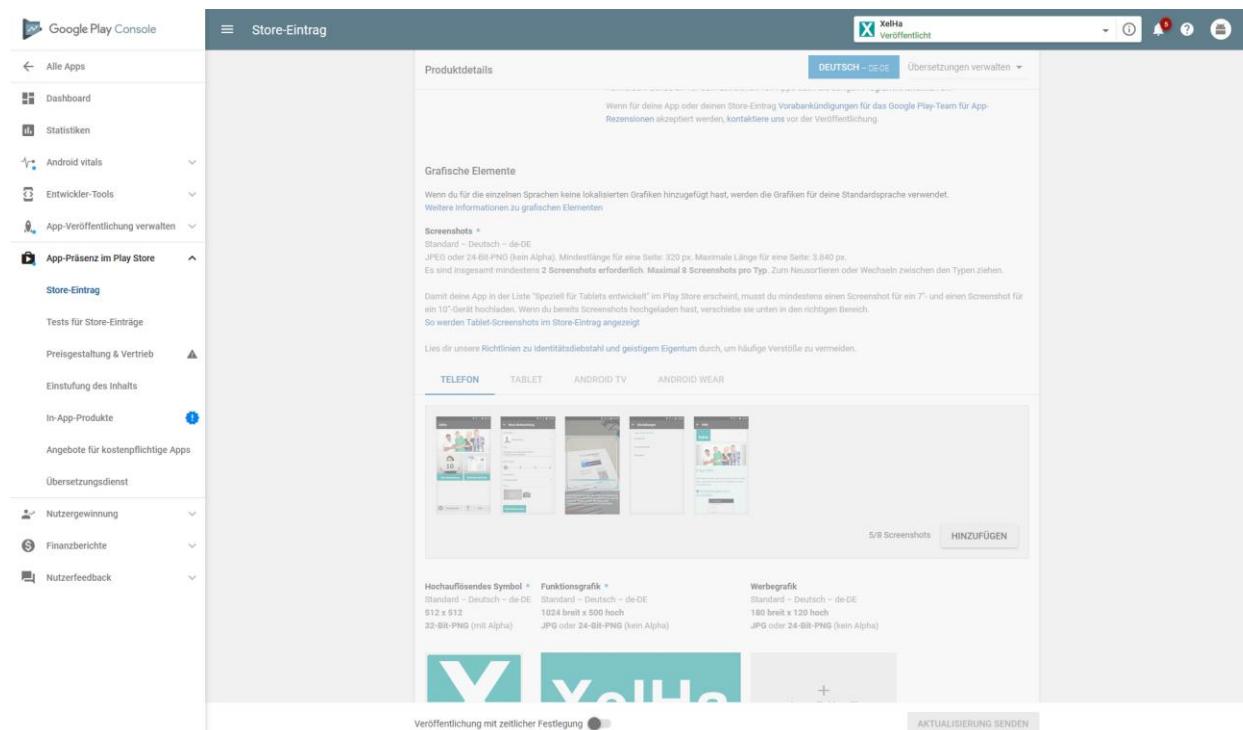
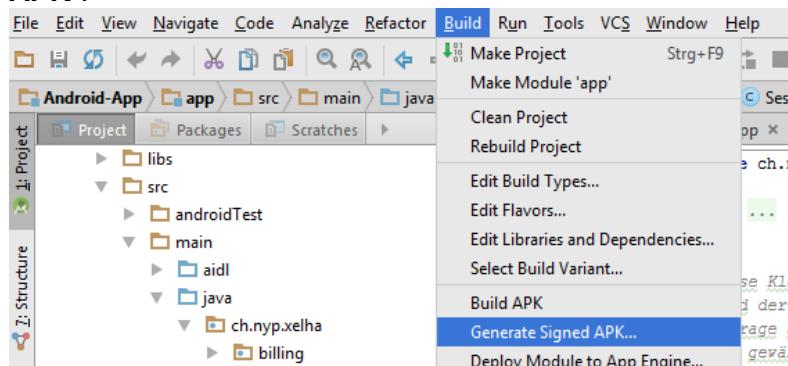


Abbildung 66 Store-Eintrag der XelHa-App

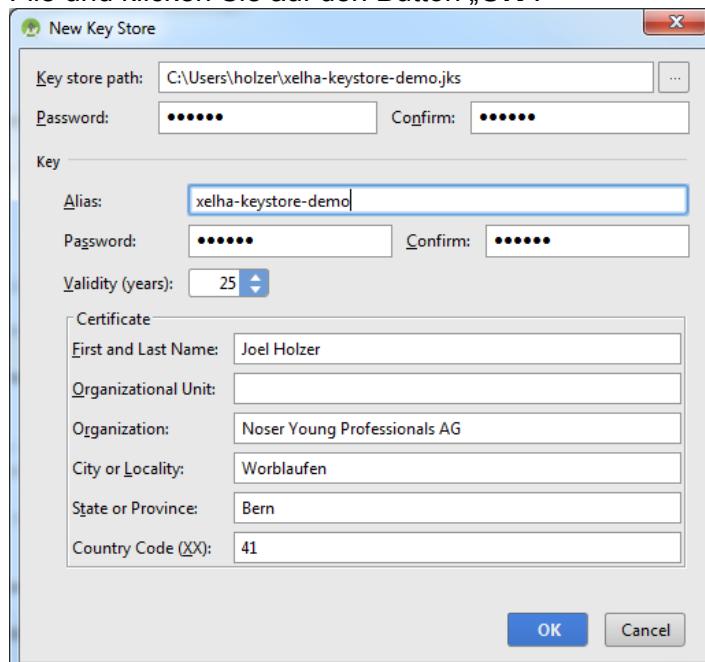
13.3 Release der App erstellen

Damit Sie die App im App-Store hochladen können, müssen Sie mit Android-Studio eine .apk-Datei der App erstellen. Folgend Sie dazu den folgenden Schritten:

1. Klicken Sie in Ihrem Projekt in Android-Studio auf das Menü „Build → Generate Signed APK“.

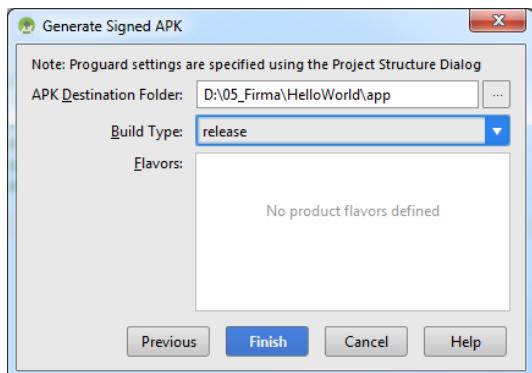


2. Nun öffnet sich ein Fenster, wo Sie Ihre App mit einem Zertifikat, dem sogenannten Keystore-File signieren müssen. Ein APK muss signiert sein, sonst kann es nicht im Google Play Store hochgeladen werden. Da Sie noch kein Keystore-File haben, müssen Sie über den Button „Create new..“ ein Keystore-File erstellen. Machen Sie nun die Angaben für Ihr Keystore-File und klicken Sie auf den Button „OK“:



3. Nun wird das erstellte Keystore-File für das generierte APK angegeben (in der vorhergehenden Maske). Klicken Sie nun auf den Button „Next“. Nun müssen Sie Ihr Master Passwort eingeben um die Passwort-Datenbank zu freizuschalten (unlocken).
4. Jetzt können Sie das APK Ihrer App generieren. Wählen Sie beim Build Type „release“ aus, denn Sie wollen ja einen Release der App erstellen.

Modul 335: Mobile-Applikationen realisieren



13.4 App in Store hochladen

13.4.1 Testphasen einer App

In der Praxis ist es nicht empfehlenswert, eine App im Google Play Store hochzuladen und direkt zu veröffentlichen. Die App sollte vor der Veröffentlichung nämlich von verschiedenen Usern auf verschiedenen Geräten ausgiebig getestet werden.

Der Google Play Store bietet Entwicklern die Möglichkeit, ihre App als Alpha-Test oder Beta-Test hochzuladen. Für den Alpha- oder Beta-Test können die Google Accounts der Tester eingetragen werden. Alle Personen, wessen Google Account eingetragen wurde, können die App im App-Store herunterladen um diese zu testen. Auch können Sie den Store-Eintrag anschauen und überprüfen ob dieser korrekt ist.

Wichtig sind Alpha- und Beta-Test vor allem bei kostenpflichtigen Apps, denn bei Alpha- und Beta-Tests können kostenpflichtige Apps kostenlos zum Testen heruntergeladen werden. Auch können die In-App-Käufe kostenlos getestet werden.

Grundsätzlich empfiehlt sich für grössere Apps das Vorgehen:

Alpha Test → Beta Test → Release

Bei kleineren Apps ist eine Testphase (Beta Test) meistens ausreichend.

Alpha und Beta Tests können unter „App-Versionen“ im Menüpunkt „App-Veröffentlichung verwalten“ in der Google Play Developer Console eingerichtet werden.

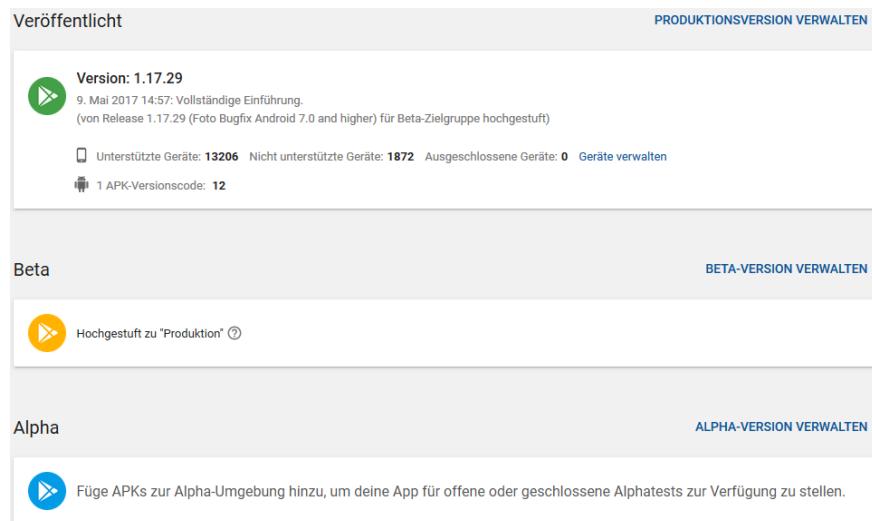


Abbildung 67 Beta-Test der XelHa-App

13.4.2 App veröffentlichen

Die App kann über den Reiter „App-Versionen“ im Menüpunkt „App-Veröffentlichung verwalten“ in der Google Play Developer Console veröffentlicht werden. Nachdem die App veröffentlicht wurde dauert es etwa einen halben Tag bis diese im Google Play Store für jedermann zugänglich ist.

13.5 App im Store updateen

Das Vorgehen um eine App im Google Play Store zu aktualisieren ist dasselbe wie beim Hochladen einer App.

Wichtig ist, dass Sie, bevor Sie für die überarbeitete App ein signiertes APK generieren, den Versions-Code Ihrer App im **build.gradle** des Moduls **app** erhöhen. Diese Nummer beginnt bei 1. Beim zweiten Release muss sie 2 betragen, usw. Wenn Sie den Version-Code nicht erhöhen, können Sie das APK nicht in den Google Play Store hochladen, da die Upload-Routine bemerkt, dass die aktuelle Version bereits über diesen Versionscode verfügt.

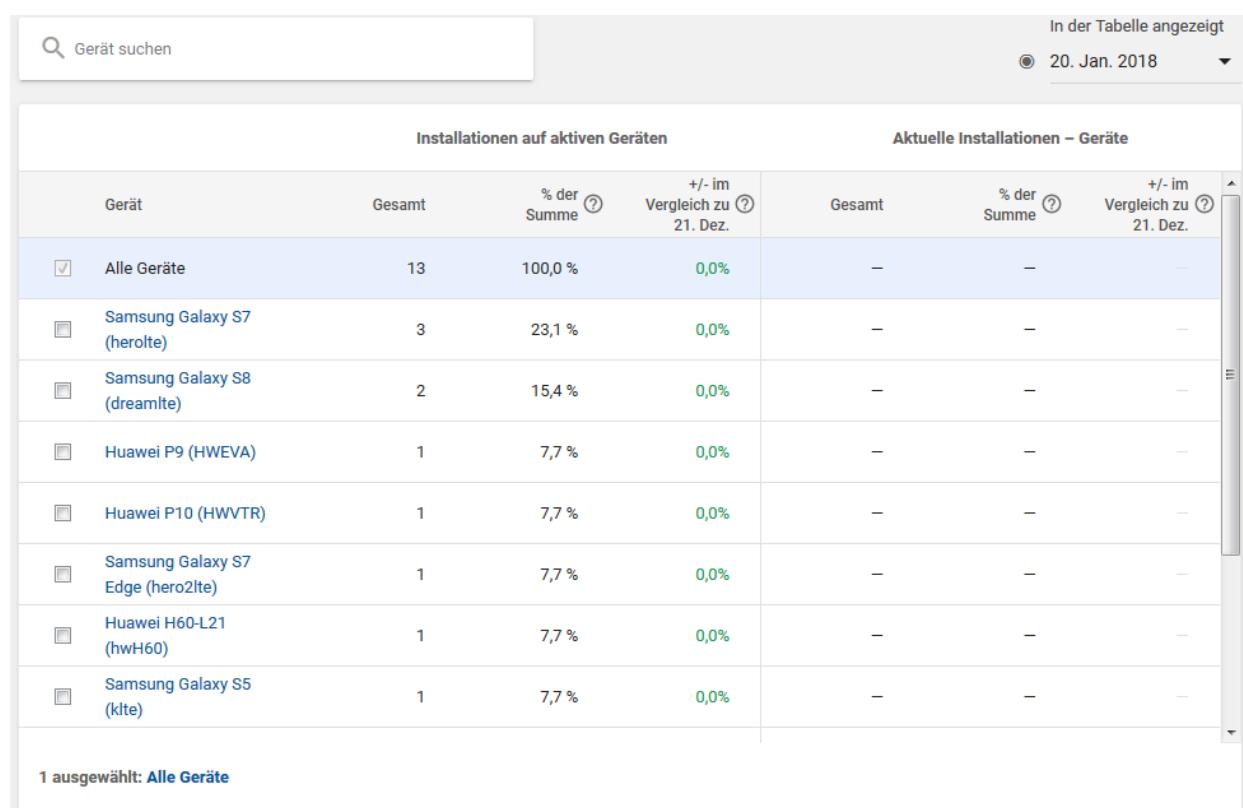
```
defaultConfig {  
    applicationId "ch.nyp.helloworld"  
    minSdkVersion 14  
    targetSdkVersion 23  
    versionCode 1  
    versionName "1.0"  
}
```

13.6 Statistiken in der Google Play Developer Console

In der Google Play Developer Console können Sie eine Vielzahl von Statistiken über Ihre hochgeladenen Apps einsehen.

Im Raster „**Statistiken**“ ist beispielsweise ersichtlich, wie viele Personen aktuell eine App installiert haben und wie der Downloadverlauf/Installationsverlauf der App aussieht. Zudem kann eingesehen werden, auf was für unterschiedlichen Android-Versionen die App installiert ist, auf was für Geräten, in welchen Ländern, in welcher Smartphone-Sprache, welche App-Version, welche Mobilfunkanbieter die Leute nutzen, und noch viel mehr.

Nachfolgende Abbildung zeigt als Beispiel die Statistik über die Gerätetypen, auf welchen die XelHa-App installiert ist.



Installationen auf aktiven Geräten				Aktuelle Installationen – Geräte			
Gerät	Gesamt	% der Summe	+/- im Vergleich zu 21. Dez.	Gesamt	% der Summe	+/- im Vergleich zu 21. Dez.	
<input checked="" type="checkbox"/> Alle Geräte	13	100,0 %	0,0%	–	–	–	
<input type="checkbox"/> Samsung Galaxy S7 (herolte)	3	23,1 %	0,0%	–	–	–	
<input type="checkbox"/> Samsung Galaxy S8 (dreamlte)	2	15,4 %	0,0%	–	–	–	
<input type="checkbox"/> Huawei P9 (HWEVA)	1	7,7 %	0,0%	–	–	–	
<input type="checkbox"/> Huawei P10 (HWVTR)	1	7,7 %	0,0%	–	–	–	
<input type="checkbox"/> Samsung Galaxy S7 Edge (hero2lte)	1	7,7 %	0,0%	–	–	–	
<input type="checkbox"/> Huawei H60-L21 (hwH60)	1	7,7 %	0,0%	–	–	–	
<input type="checkbox"/> Samsung Galaxy S5 (klte)	1	7,7 %	0,0%	–	–	–	

1 ausgewählt: Alle Geräte

Abbildung 68 Beispiel einer Statistik in der Google Play Developer Console

In den Tabs „Nutzergewinnung“ und „Bewertung & Rezensionen“ können Sie zudem weitere Statistiken über Ihre App, sowie die Bewertungen und Kommentare zu Ihrer App, einsehen.

Diese detaillierten Statistiken können sehr hilfreich sein bei der Optimierung und Vermarktung der App.

14 Abbildungsverzeichnis

Abbildung 1: XelHa Native-App für Android	10
Abbildung 2 „Hello World“-Programm mit Swift und Objective-C [1].....	11
Abbildung 3 Storyboard von XCode [4].....	12
Abbildung 4 Unterschied Web App und normale Webseite am Beispiel von immowelt.de	13
Abbildung 5 Vergleich des Aufbaus von Native Apps, Web Apps und Hybrid Apps [2]	15
Abbildung 6: Vergleich von Native, Web und Hybrid App in Bezug auf mögliche Kundenanforderungen [3].....	17
Abbildung 7 Cross Plattform Entwicklung mit Xamarin [5]	18
Abbildung 8 Marktanteil Mobile Betriebssysteme in der Schweiz [6].....	19
Abbildung 9 Marktanteil Mobile Betriebssysteme in Europa [7].....	19
Abbildung 10 Systemarchitektur der Android Plattform [8].....	22
Abbildung 11 Android SDKs installieren	24
Abbildung 12 AVD Manager	25
Abbildung 13 Neues virtuelles Gerät erstellen (Hardware Profil auswählen).....	26
Abbildung 14 Hardware-Profil für Samsung Galaxy S6 erstellen	27
Abbildung 15 build.gradle der XelHa-App.....	28
Abbildung 16 HelloWorld-App.....	29
Abbildung 17 Neues Projekt erstellen – Projektname angeben	29
Abbildung 18 Neues Projekt erstellen - Activity Name und Layout Name	30
Abbildung 19 Projektstruktur „HelloWorld“-App	31
Abbildung 20 activity_main.xml der HelloWorld-App.....	32
Abbildung 21 Manifest-Datei der HelloWorld-App	34
Abbildung 22 HelloWorld-App im Emulator ausführen	35
Abbildung 23 HelloWorld-App auf Smartphone übertragen	35
Abbildung 24 Breakpoint in der Klasse „MainActivity“	36
Abbildung 25 HelloWorld-App debuggen.....	36
Abbildung 26 HelloWorld-App debuggen 2	36
Abbildung 27 Beispiele von GUI-Komponenten mit Material Design. [8].....	40
Abbildung 28 Activity mit XML-Datei im Text- und Designmodus, sowie Java-Code der Activity	41
Abbildung 29 Starten und Beenden von Activities am Beispiel der XelHa-App	42
Abbildung 30 List-Detail-Ansicht von Smartphone und Tablet mit Hilfe von Fragments	44
Abbildung 31 Navigation Drawer [9]	45
Abbildung 32 GUI-Designer für Realisierung von Activity/Fragment	46
Abbildung 33 XML-Ansicht für Realisierung von Activity/Fragment.....	47
Abbildung 34 horizontales LinearLayout	48
Abbildung 35 RelativeLayout.....	48
Abbildung 36 ConstraintLayout.....	48
Abbildung 37 Android-Geräte mit verschiedenen Bildschirmgrößen und Auflösungen [11]	52
Abbildung 38 Beispiel von Bilder für unterschiedliche generalisierte Pixeldichten.....	53
Abbildung 39 Quer- und Hochformat einer App, welche Fragments verwendet (Nexus 9 Tablet)	54
Abbildung 40 Ausrichtungsspezifische Layout-Files	55
Abbildung 41 Gegenüberstellung Code 1 Fragment und 2 Fragment in einer Activity anzeigen	55
Abbildung 42 Activity Lifecycle [12].....	58
Abbildung 43 Fragment Lifecycle	60
Abbildung 44 Demo-App zum Verarbeiten von Benutzereingaben (ActionListener).....	63
Abbildung 45 Expliziter Intent, welcher eine neue Activity startet [13].....	70
Abbildung 46 Demo App zum Starten einer neuen Activity mit einem Intent.....	71

Abbildung 47 Beispiele von Dialogen [14]	74
Abbildung 48 Dialog, welcher erstellt wird	74
Abbildung 49 Spinner [14]	75
Abbildung 50 Spinner mit dynamischen Einträgen und eigenem Layout.....	77
Abbildung 51 ListView mit dynamischen Einträgen und eigenem Layout.....	78
Abbildung 52 Shared Preferences sind optimal für die Speicherung von App-Einstellungen geeignet.	79
Abbildung 53 Objektrelationales Mapping [16].....	87
Abbildung 54 Beispiel eines verteilten Systems mit Webservices (Wetterstation) [17].....	91
Abbildung 55 REST-Service, oft auch REST API genannt [18].....	94
Abbildung 56 Postman Google Chrome Plugin.....	95
Abbildung 57 Fingerabdrucksensor bei einem iPhone [18]	96
Abbildung 58 Sensor-Demo-App – Luftdruck anzeigen	97
Abbildung 59 Generierung einer Random-Nummer wird in eigenen Thread ausgelagert.....	100
Abbildung 60 White Box Testing [20].....	103
Abbildung 61 Unit-Test-Verzeichnisse in App-Projekt.....	105
Abbildung 62 Lokale Unit-Tests Paketstruktur	105
Abbildung 63 Instrumented Unit-Tests Paketstruktur.....	108
Abbildung 64 Black Box Testing [21]	110
Abbildung 65 Icon vom Google Play Store	116
Abbildung 66 Store-Eintrag der XelHa-App	117
Abbildung 67 Beta-Test der XelHa-App.....	119
Abbildung 68 Beispiel einer Statistik in der Google Play Developer Console.....	121

15 Tabellenverzeichnis

Tabelle 1 Änderungsgeschichte	3
Tabelle 2 Code-Beispiele / Demo-Projekte auf GitHub	9
Tabelle 3 Dateien des Value-Verzeichnis	33
Tabelle 4 Beispiel-Testfall	112
Tabelle 5 Geräte pro Bildschirmklasse	112

16 Quellen

- [1] **Abbildung Swift Objective C**
<http://de.slideshare.net/wevtimoteo/swift-language-a-fast-overview>

- [2] **Abbildung Hybrid App**
<http://www.innotix.com/blog/hybrid-mobile-apps-hype-oder-trend>

- [3] **Abbildung Vergleich App-Typen**
<http://digitale-exzellenz.de/eine-digital-exzellente-frage-native-hybrid-oder-web-app/>

- [4] **Abbildung Storyboard von XCode**
https://cdn.tutsplus.com/mobile/uploads/legacy/iOS-SDK_Storyboards/StoryboardEditorZoomedOut.png

- [5] **Abbildung Xamarin**
<http://www.slideshare.net/Xamarin/introduction-to-xamarinform>

- [6] **Statistik Marktanteile mobile Betriebssysteme Schweiz**
<https://www.tagesanzeiger.ch/wirtschaft/standard/iPhone-ist-nicht-mehr-beliebtestes-Smartphone-der-Schweiz/story/25155855>

- [7] **Statistik Marktanteile mobile Betriebssysteme Europa**
<https://de.statista.com/statistik/daten/studie/184516/umfrage/marktanteil-der-mobilien-betriebssysteme-in-europa-seit-2009/>

- [8] **Systemarchitektur Android-Plattform**
<https://developer.android.com/guide/platform/index.html>

- [9] **Abbildung Material Design**
https://de.wikipedia.org/wiki/Material_Design

- [10] **Abbildung Navigation Drawer**
<http://www.androidhive.info/2013/11/android-sliding-menu-using-navigation-drawer/>

- [11] **Abbildung verschiedene Bildschirmgrößen**
<https://www.inserteffect.com/blog/designs-fuer-android-erstellen>

- [12] **Abbildung Activity Lifecycle**
<http://www.herongyang.com/Android/Android-Application-Activity-Lifecycle.jpg>

- [13] **Abbildung Intent**
<http://www.vogella.com/tutorials/AndroidIntent/article.html>

- [14] **Abbildung Dialoge**
<https://developer.android.com/guide/topics/ui/dialogs.html>

- [15] **Abbildung Spinner**
<https://developer.android.com/guide/topics/ui/controls/spinner.html>

- [16] **Abbildung ORM**
<http://www.agile-code.com/blog/microsoft-net-or-mapper-choose-your-own/>

- [17] **Abbildung Webservice Wetterstation**
<https://www.toptal.com/c/how-i-made-a-fully-functional-arduino-weather-station-for-300>

- [18] **Abbildung REST-Service**
<https://geekli.st/khaledisamm/links/12142>

- [19] **Abbildung Fingerabdrucksensor**
http://www.t-online.de/handy/smartphone/id_65596510/iphone-5s-hacker-sollen-fingerabdruck-sensor-ueberlisten.html

- [20] **Abbildung White Box Testing**
<http://jobsandnewstoday.blogspot.ch/2013/03/what-is-white-box-testing.html>

- [21] **Abbildung Black Box Testing**
<http://www.softwaretestingsoftware.com/wp-content/uploads/2010/04/blackbox-testing.jpg>
