Technik für Kinder e.V.
Begeistern durch Machen

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Modul:

App-Entwicklung mit Java Android

Java

Version: 3.0

Dieses Modul wurde an der Lehr- und Forschungseinheit Mathematik und Informatik (LMI) der Universität Passau in Kooperation mit dem Verein „TfK - Technik für Kinder e.V.“ im Rahmen des Projekts Technikgruppen an Schulen - Kinder für Technik begeistern erarbeitet.
Mehr zu diesem Projekt erfahren Sie auf der Internetseite

http://www.fim.uni-passau.de/fim/fakultaet/lehrstuehle-professuren-und-fachgebiete-der-fim/didaktik-der-informatik/projekte/technikgruppen.html .

Autor:

Wolfgang Pfeffer

Didaktik der Informatik / Mathematik
wolfgang.pfeffer@uni-passau.de

Modulübersicht

Im Rahmen des Kooperationsprojekts zwischen der Universität Passau und dem Verein Technik für Kinder e.V. sind folgende Module entstanden:

Modul Roboter-Programmierung

Dieses Modul bietet einen sehr kurzen Einstieg in die Programmierung eines Lego EV3 Roboters mit der Lego Mindstorms Education Software. Neben Hardware- und Software Anforderungen wird die Entwicklungsumgebung vorgestellt und anschließend Einstiegsaufgaben zu Motoren, Sensoren sowie vermischte Aufgaben bereitgestellt.

Modul App-Entwicklung auf Mobile Devices

Dieses Modul thematisiert umfassend die App-Entwicklung auf Mobile Devices mit dem AppInventor, einer graphischen Programmierumgebung. Ein einfacher und handlungsorientierter Zugang zu dieser Thematik steht dabei im Vordergrund. Neben Hardware- und Softwareanforderungen wird eine ausführliche Installationsanleitung angeboten. Die Einführungsaufgabe 'Schritt für Schritt zur ersten eigenen App' ermöglicht eine Einführung in den Umgang mit dem AppInventor. Gleichzeitig werden fast alle wichtigen Elemente behandelt. Anschließend umfasst das Modul 10 Aufgaben mit unterschiedlichem Schwierigkeitsgrad sowieso ausführlichen Lösungen. Exkurse zu Themen wie GPS und Dijkstra-Algorithmus runden die Handreichung ab.

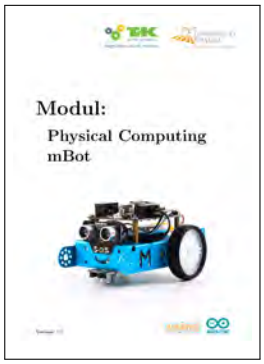
Modul App-Entwicklung mit Java Android

Dieses Modul baut thematisch auf dem Modul 'App-Entwicklung auf Mobile Devices' auf, anstelle der graphischen Programmierung werden die Applikationen nun mit Java Android entwickelt. Da die Einstiegshürde in Java Android vergleichsweise hoch ist, sind grundlegende Java-Kenntnisse unumgänglich. Es wird ein sehr ausführlicher sowie kleinschrittiger Einstieg in die Programmierung mit Java Android geboten. Anschließend stehen verschiedene Projekte mit umfangreichen Lösungen zur Auswahl (z.B. Vokabel-App, Quiz-App, 2048-App, Datenspeicher oder LegoPilot-App). Dabei werden wichtige Konzepte der Sekundarstufe II behandelt.

Modul Physical Computing

Dieses Modul kombiniert Elemente aus der Physik und der Informatik und basiert auf dem Einplatinencomputer Raspberry Pi. Zunächst geht es darum, (einfache) Schaltungen zu erstellen und mit konstanter Spannungsquelle zu arbeiten. Hierbei wird mit verschiedenen Schaltungselementen experimentiert (z.B. LED, Widerstand, LDR, Poti, Servomotor). Anschließend soll an manchen Pins gezielt Spannung angelegt oder nicht angelegt werden. Dazu werden die GPIO-Pins mit der Programmiersprache Python konfiguriert und angesteuert. Die Handreichung beinhaltet ein eigenes Kapitel zu Python, in welchem man sich anhand kleiner Aufgaben mit der Syntax der Programmiersprache vertraut machen kann.

Modul Physical Computing mBot



Dieses Modul kann als Schnittstellenmodul zwischen Roboter-Programmierung und Physical Computing gesehen werden. Bei mBot handelt es sich um ein handliches Roboterfahrzeug, das mit verschiedenen Sensoren und Motoren ausgestattet ist, die mit dem Mikrocontroller Arduino verbunden werden können. Im Bezug auf die Programmierung bietet das Modul zwei Zugangswege: Ähnlich zur App-Entwicklung mit dem AppInventor kann der mBot mithilfe der graphischen Programmierumgebung Scratch angesteuert werden. Ein zweiter Teil des Moduls bietet die Programmierung des mBot mit Hilfe von Arduino-C.

Quelle des Bildes: https://www.colourbox.de/vektor/buch-vektor-5814145

Inhaltsverzeichnis

Kapitel 1

Über das Modul

App-Entwicklung mit Java Android - Wie der Titel schon vorwegnimmt, beschäftigen wir uns in diesem Modul mit der Entwicklung von eigenen Applikationen für Smartphone und Tablet. Aktuellen Studien zu Folge besitzen rund 80 % der Kinder und Jugendlichen ein eigenes Smartphone. Das Nutzen von Apps wie Nachrichtendienste, Spiele oder Internetbrowser ist selbstverständlich geworden. Doch was steckt eigentlich hinter einer App? Wie entsteht eine App? Wie aufwändig ist es, eine eigene App zu erstellen?
Diesen Fragen möchten wir in diese Modul auf den Grund gehen.

In dem Modul „App-Entwicklung auf Mobile Devices“ stand vor allem ein einfacher und handlungsorientierter Zugang zu dieser Thematik im Vordergrund. Wir haben dort den AppInventor von MIT (Massachusetts Institute of Technology) verwendet, mit dem wir unabhängig von einer konkreten Programmiersprache und -syntax durch Zusammenfügen von vorgefertigten Bausteinen Android-Apps erstellen konnten.

In diesem Modul tauchen wir nun tiefer in die Thematik ein und werden die Apps mit der Programmiersprache Java-Android erstellen. Die Einstiegshürde ist deutlich höher und es ist sehr empfehlenswert, dass die Schülerinnen und Schülern grundlegende Java-Kenntnisse mitbringen. Hat man die Einstiegshürde überwunden, stehen einem alle Türen offen. Während der AppInventor die Möglichkeiten noch begrenzte (es konnten z.B. keine eigenen Klassen definiert werden), sind einem bei Java-Android keine Grenzen mehr gesetzt.

In Kapitel 2 haben wir Ihnen die benötigte Hardware und Software aufgelistet, um mit diesem Modul arbeiten zu können. In Kapitel 3 zeigen wir Ihnen, wie Sie ihr Smartphone oder Tablet vom Computer als ADB Interface erkennen lassen können. So richtig los gehts es dann ab Kapitel 4 mit einer Einführungsaufgabe, in der wir auf die wichtigsten Komponenten der App-Entwicklung mit Java-Android eingehen. In den Kapiteln 5 - 8 werden Sie dann Ihre ersten eigenen Applikationen erstellen, wobei die Aufgabenstellungen so konzipiert sind, dass Sie mehr und mehr selbstständig arbeiten werden.

Bitte senden Sie Ihre Erfahrungen, Anregungen oder Fragen zu diesem Modul an wolfgang.pfeffer@uni-passau.de .

Kapitel 2

Was wird benötigt?

Überblick:

Zunächst möchten wir auf die technischen Voraussetzungen eingehen. Wir unterscheiden dabei erforderliche Hardware und erforderliche Software:

2.1 Hardware

Jeder Arbeitsplatz benötigt ein Smartphone bzw. Tablet mit einem Android-Betriebssystem. Für iPhone, iPad und Mobile Devices mit anderen Betriebssystemen (Windows, o.ä.) werden wir keine Apps entwickeln. Neben dem Mobile Device wird noch ein USB-Kabel benötigt, damit man es mit einem Laptop bzw. Desktop-PC verbinden kann:

Abbildung 2.1: Mobile Device mit Android-Betriebssystem und USB-Verbindungskabel

Zusätzlich benötigen wir natürlich einen Laptop oder Desktop-PC mit Internetzugang.

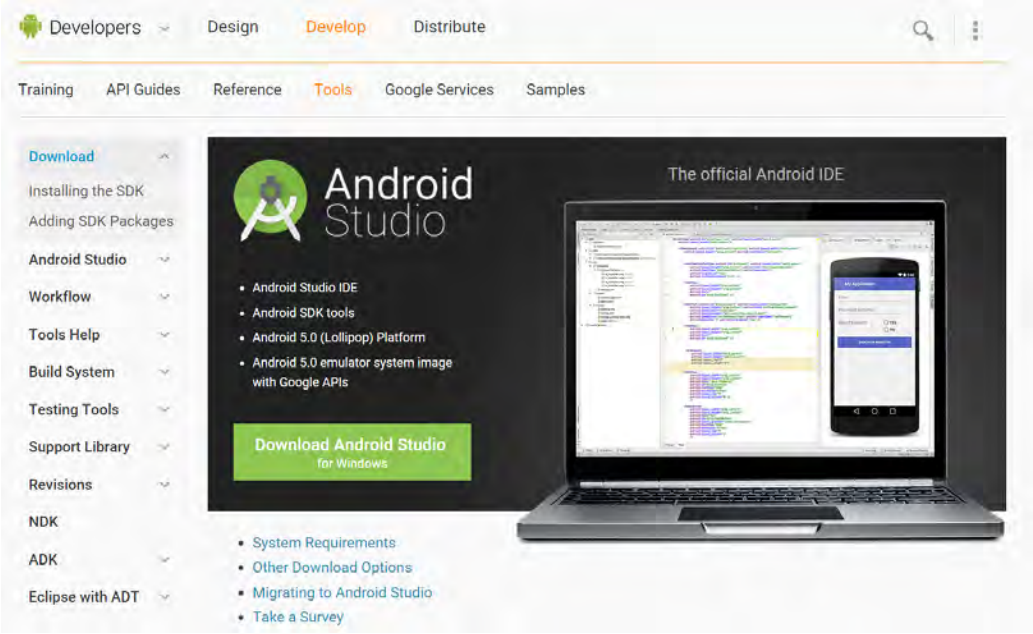
2.2 Software

Auf dem Laptop bzw. Desktop-PC ist die Software Java erforderlich. Ist Java auf ihrem Laptop bzw. Desktop-PC nicht installiert, können Sie die Software unter folgendem Link herunterladen:

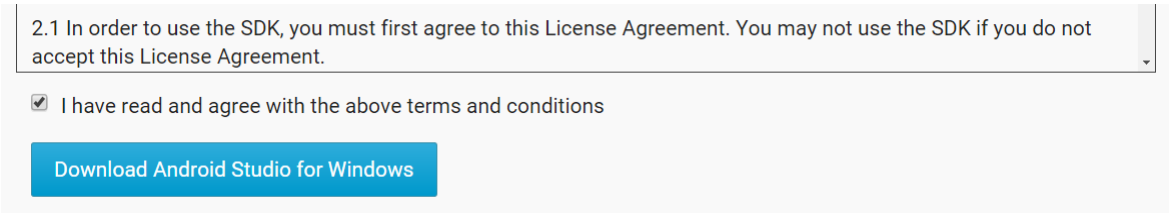
`http://www.java.com/de/download/`

Als Entwicklungsumgebung verwenden wir das auf `IntelliJ` basierende Android Studio, die offizielle Entwicklungsumgebung für Android. Zusätzlich brauchen wir noch die Android SDK Tools. Das Komplettpaket können Sie auf folgender Seite herunterladen:

`https://developer.android.com/sdk/index.html`



Stimmen Sie den Lizenzbedingungen zu und laden Sie die Installationsdatei herunter:



Installieren Sie Android Studio und die Android SDK Tools auf Ihrem Rechner.

Kapitel 3

Vorbereitung - Mobile Device als ADB Interface erkennen

Wir verweisen an dieser Stelle auf die Abschnitte 3.5 und 3.6 im Modul „App-Entwicklung auf Mobile Devices“. Hier sind die gleichen Schritte erforderlich.

Kapitel 4

Einführungsaufgabe - erste Schritte mit Java Android

Überblick:

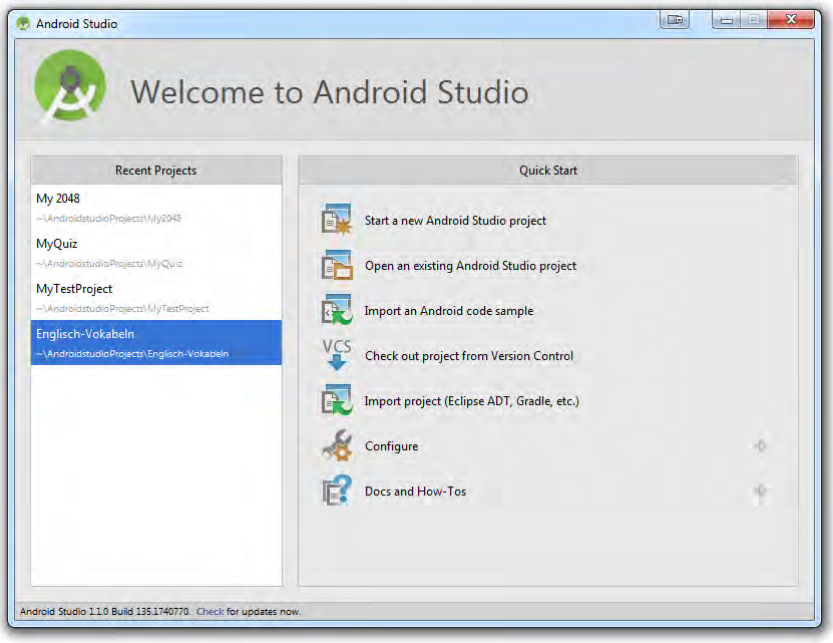
In diesem Kapitel erstellen wir eine einfache App, bei der man über eine Schaltfläche die Hintergrundfarbe des Mobile Device ändern kann. Anschließend sollen wir der Schaltfläche über ein Textfeld eine neue Beschriftung zuweisen können.
Ziel ist es, einen ersten Einblick in Aufbau und Funktionsweise eines Android-Applikations-Projekts zu erhalten.

4.1 Projekt anlegen

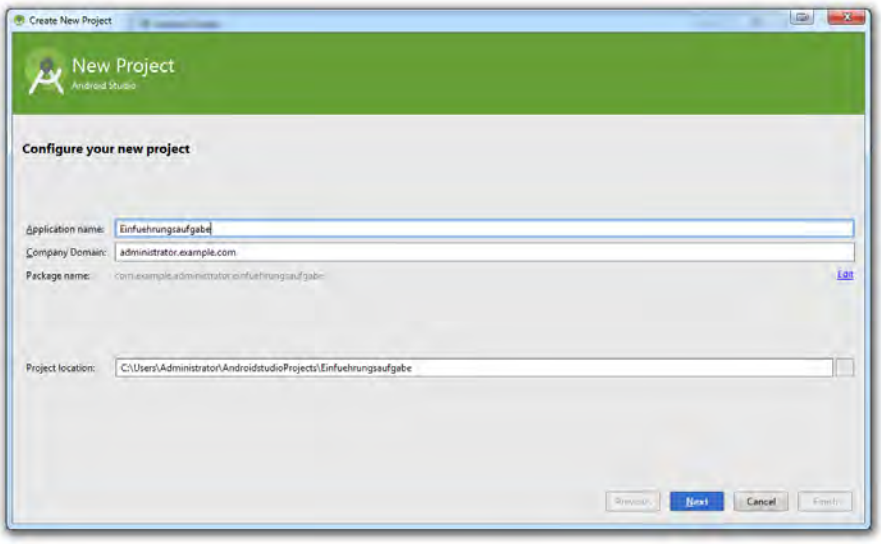
Wir gehen im Folgenden davon aus, dass Android Studio und die Android SDK Tools installiert wurden (sonst s. Abschnitt 2.2) und das Mobile Device vom Computer als ADB Interface erkannt wird.

Wir starten Android Studio

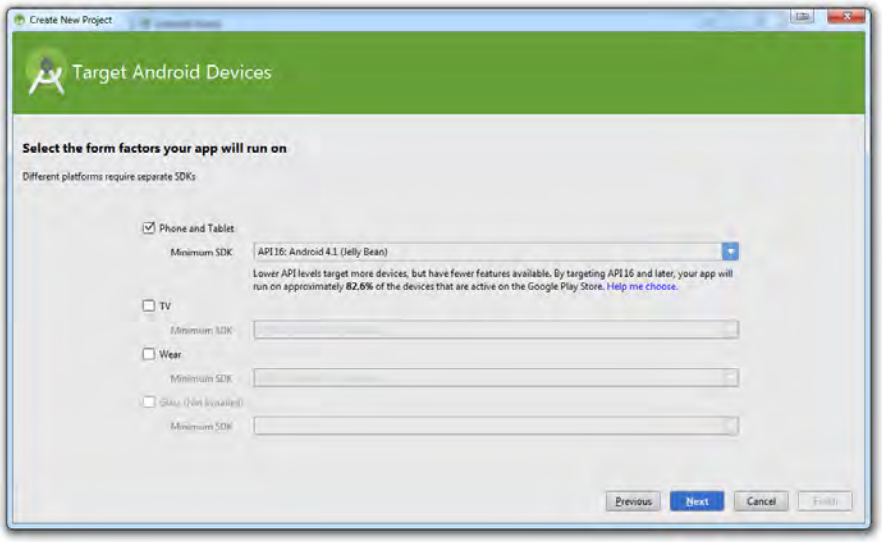
und wählen die Option `Start a new Android Studio project`:



Es öffnet sich ein Fenster mit dem Namen `Create new Project`, in dem wir den Namen unserer App eingeben können. Wir verwenden den Namen *Einfuehrungsaufgabe*:

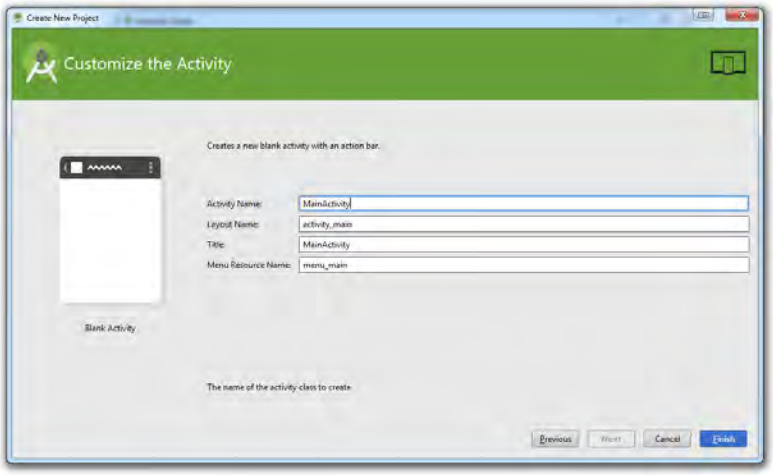


Im nächsten Schritt müssen wir angeben, für welche Geräte wir die App entwickeln wollen. Wir wählen hier die Option `Phone and Tablet` und als Minimum Required SDK werden wir `API 16: Android 4.1 (Jelly Bean)`:

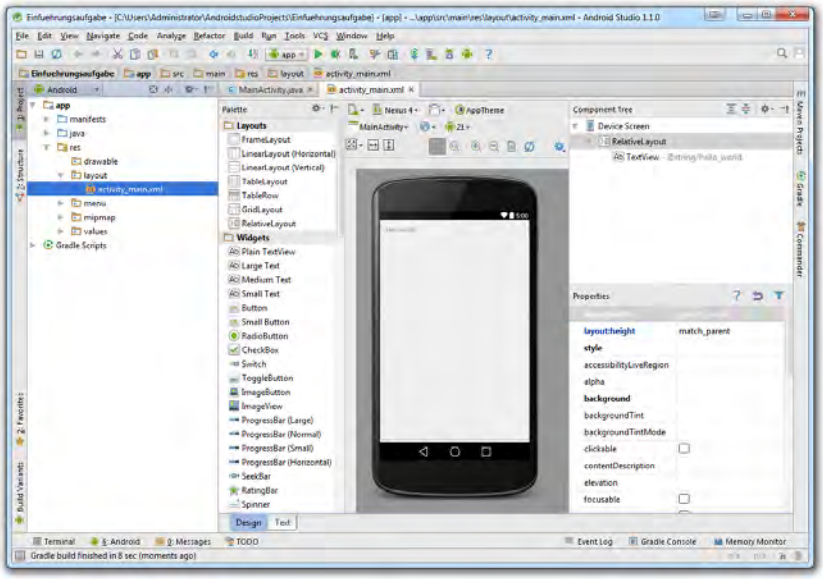


Im nächsten Schritt können wir angeben, welche Art von Activity-Klasse (dazu später mehr) wir bereits automatisch zu unserem Android Application Projekt hinzufügen wollen. Wir wählen hier die Option `Blank Activity`:

Im nächsten Schritt können wir der Activity-Klasse einen Namen geben. Wir behalten wir die Standardeinstellungen bei und beenden das Menü mit `Finish`:

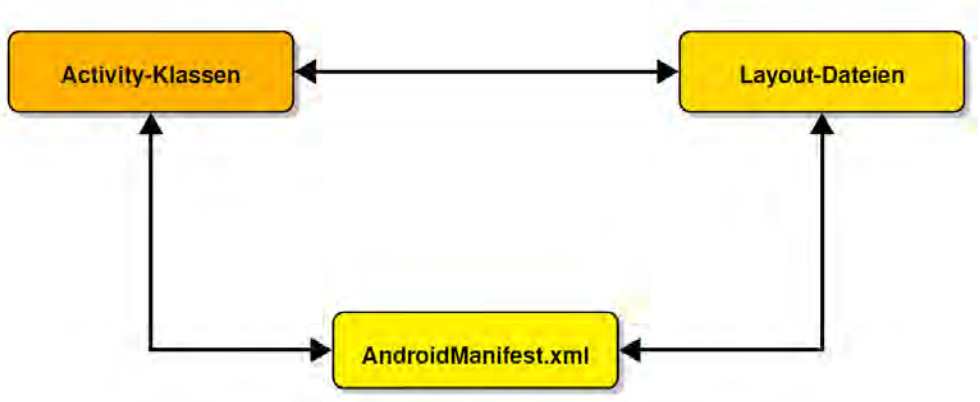


Nach kurzer Wartezeit öffnet sich ein neues Fenster mit der Übersicht über unser Android Applikation Projekt:



4.2 Die Komponenten

Für uns sind grundsätzlich drei verschiedene Bereiche des Projekts wichtig:



- **Activity-Klassen**

  Eine Activity kombiniert Bildschirmseite und Code. Man kann jeder Activity-Klasse eine oder auch unterschiedliche Layout-Dateien zuweisen. In der Activity-Klasse legt man z.B. fest, wie auf Benutzerinteraktion (z.B. Drücken einer Schaltfläche) reagiert werden soll.

- **Layout-Datei**

  Jeder Activity-Klasse weisen wir eine Layout-Datei zu. In dieser können wir über einen grafischen Editor Objekte anlegen und initialisieren.

- **AndroidManifest.xml**

  Diese Datei beinhaltet wichtige Informationen über die App für das Android-System (z.B. minimale SDK-Version, maximale SDK-Version, etc.). Zudem muss jede neue Activity-Klasse in dieser Datei eingetragen werden. Ansonsten treten beim kompilieren Fehler auf.

4.2.1 AndroidManifest.xml

In der linken Leiste findet man den Ordner `manifests`, in dem sich die Datei `AndroidManifest.xml` befindet. In dieser Datei befinden sich essentielle Informationen der App für das Android-System, auf dem sie laufen soll. Neben den Namen der App werden noch weitere wichtige Informationen, wie etwa die Namen aller Activity-Klassen, sowie Berechtigungen (z.B. Zugriff auf das Internet), in dieser Datei gespeichert. Erstellen wir eine neue Activity-Klasse, wird diese hier automatisch eingetragen.

In unserem Fall bekommen wir folgenden `XML`-Code zu sehen:

Code 4.1: AndroidManifest.xml

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

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

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

</manifest>
```

In diesem `XML`-Code finden wir auch unsere erstellte ActivityKlasse `MainActivity`. Wir möchten, dass unsere App nur im Hochformat ausgeführt werden kann. Dazu ergänzen wir innerhalb des Activity-Tags folgende Zeile im `XML`-Code:

```
android:configChanges="orientation"
android:screenOrientation="portrait"
```

Die `XML`-Klasse sollte nun wie folgt aussehen:

Code 4.2: activity_main.xml

```
...
        <activity
            android:name=".MainActivity"
            android:configChanges="orientation"
            android:screenOrientation="portrait"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

4.2.2 Die Klasse `MainActivity.java`

Als nächstes betrachten wir die Klasse `MainActivity.java`. Diese befindet sich im Ordner `java`. Hier wurden bereits einige Methoden automatisch generiert. Der Übersichtlichkeit halber löschen wir alle überflüssigen Methoden und behalten nur die Methode `onCreate(...)` bei:

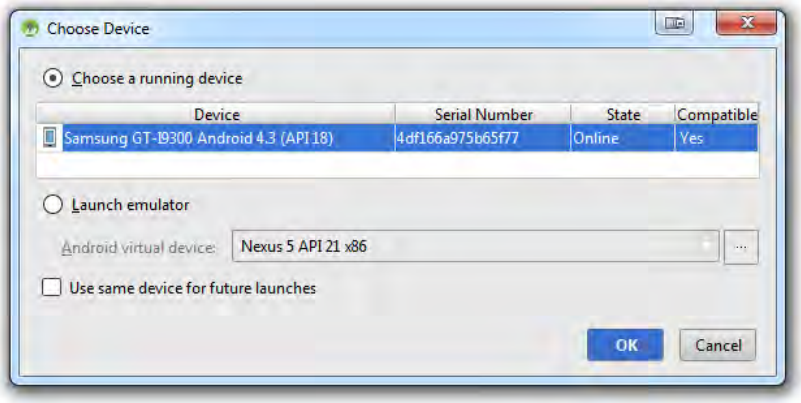
Code 4.3: MainActivity.java

```
package com.example.englisch_vokabeln;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

public class MainActivity extends ActionBarActivity {

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

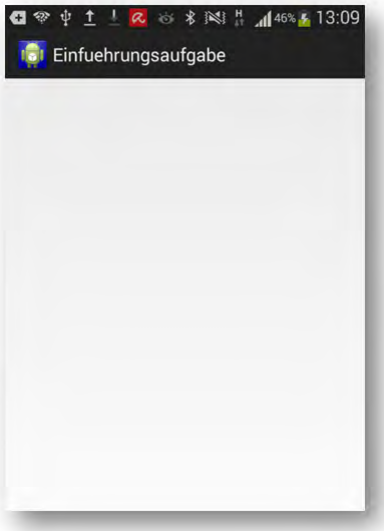
In der `onCreate(...)`-Methode findet sich der Befehl `setContentView(R.layout.activity_main)`. Hier wird der Activity-Klasse eine Layout-Datei zugewiesen.

Was passiert, wenn wir unsere App starten, ohne weitere Änderungen vorzunehmen? Dazu verbinden wir unser Mobile Device mit dem Computer, wählen in Eclipse die Datei `MainActivity.java` aus und drücken auf das  - Symbol.
Wurde unser Mobile Device vom Computer als ADB-Interface erkannt, wird es in der Liste angezeigt. Wir wählen es aus und drücken auf `Ok`[1]:

[1] Wird das Mobile Device nicht angezeigt, wurde der ADB-Treiber nicht richtig installiert. Wir verweisen hier auf Kapitel 3. Alternativ kann man die Applikation auch auf einem virtuellen Emulator ausführen (s. Abbildung)



Nach kurzer Wartezeit wird die Applikation auf dem Handy angezeigt:



Beim Starten der Applikation wird die `onCreate`-Methode der Klasse `MainActivity.java` aufgerufen:

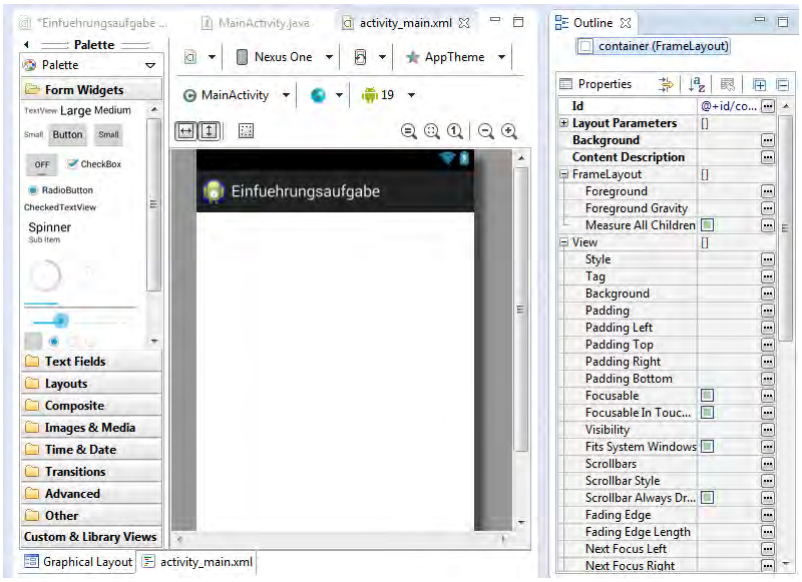
Code 4.4: onCreate-Methode der Klasse MainActivity.java

```
1 protected void onCreate(Bundle savedInstanceState) {
2     super.onCreate(savedInstanceState);
3     setContentView(R.layout.activity_main);
4 }
```

und dabei der Befehl `setContentView(R.layout.activity_main)` ausgeführt. `activity_main` ist eine `XML`-Datei, in der die graphische Oberfläche festgelegt wird und Komponenten (TextView, Button, etc.) erstellt werden können.

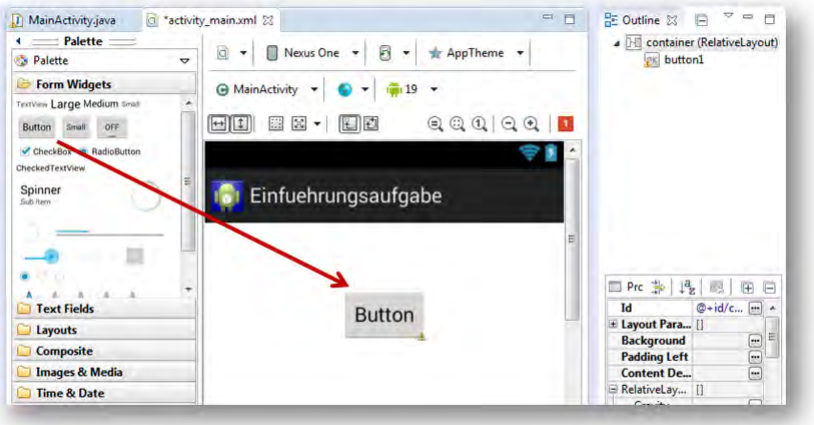
4.2.3 Die Layout-Datei `activity_main.xml`

Diese Layout-Datei findet sich im `Package-Explorer` unter `res` → `layout`.
Öffnen wir diese Datei, wird uns ein virtueller Bildschirm angezeigt:



Wir ändern zunächst das Layout des Bildschirms von `FrameLayout` auf `RelativeLayout`[2]. Dazu klicken wir mit der rechten Maustaste rechts oben auf `container (FrameLayout)` und anschließend auf `Change Layout`. Hier wählen wir das `RelativeLayout` aus.

Anschließend fügen wir mit Drag&Drop eine neue Schaltfläche auf unseren virtuellen Bildschirm hinzu:



Für die Layout-Dateien stehen uns zwei verschiedene Ansichten zur Verfügung. Einmal die graphische Ansicht (*Graphical Layout*) und einmal die XML-Ansicht (*activity_main.xml*), in der wir die Layout-Datei mit XML-Code bearbeiten können. Zwischen diesen beiden Ansichten kann man durch die zwei Tabs mit den entsprechenden Beschriftungen hin und her schalten.

Jedes graphische Element besitzt eine `ID`-Nummer. Die `ID`-Nummer können wir ganz einfach in der XML-Ansicht ändern. Es empfiehlt sich, immer aussagekräftige `ID`s zu vergeben und diese eindeutig zu halten.

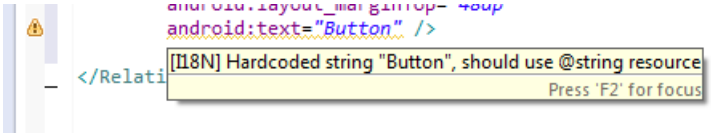
Da wir mit unserem Button-Objekt die Hintergrundfarbe ändern möchten, geben wir ihm die `ID` *changeColorButton*:

[2] Falls nicht sowieso schon das `RelativeLayout` ausgewählt ist.

```
<Button
    android:id="@+id/changeColorButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="48dp"
    android:text="Button" />
```

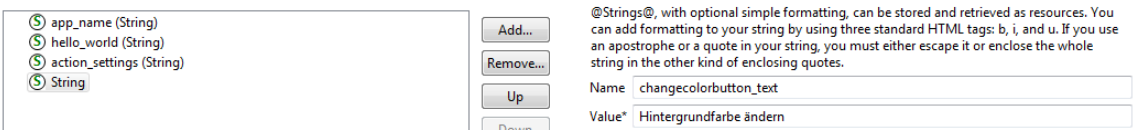
4.3 Warnmeldung bei Verwendung hardkodierter Strings

In der XML-Ansicht wird folgende Warnmeldung angezeigt:

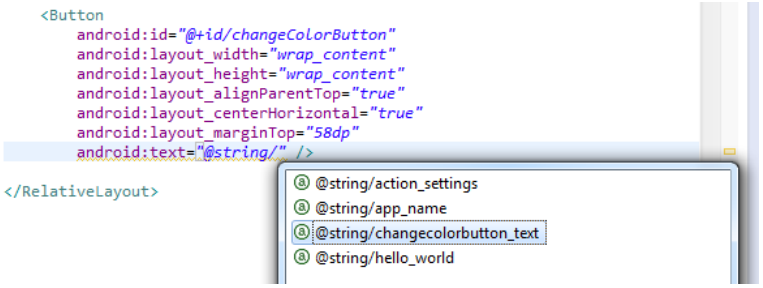


Zur besseren Wartung empfiehlt es sich, alle Strings nicht „hart zu kodieren“, sondern in einer separaten Datei zu verwalten. Möchte man nun bestimmte Beschriftungen von Schaltflächen oder ähnlichem ändern, muss man nicht alle XML-Dateien durchsuchen, sondern kann dies in einer zentralen Datei tun. Ein weiterer Vorteil besteht darin, wenn man die App beispielsweise mehrsprachig anbieten möchte und bei Sprachänderung einfach die zentrale Datei austauscht.

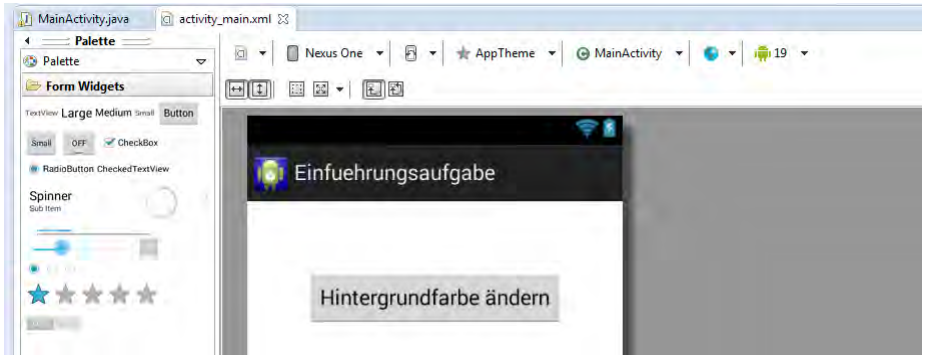
Die zentrale Verwaltung von Strings kann man in der Datei `strings.xml` im Ordner `values` vornehmen. Über die Schaltfläche `Add` legt man einen neuen String an. Wir geben ihm den Namen `changecolorbutton_text` und den Wert `Hintergrundfarbe ändern`:



Diesen String können wir nun unserem Button-Objekt in der `xml`-Ansicht der Layout-Datei zuweisen:



In der Grafik-Ansicht wird nun der entsprechende String angezeigt:



4.4 Ereignisverarbeitung

Wir haben nun in der Layout-Datei ein Button-Objekt erstellt, jedoch passiert aktuell noch gar nichts, wenn wir auf dieses Klicken. Wir möchten gerne in der Activity-Klasse eine Methode erstellen, die immer automatisch aufgerufen wird, falls auf unserer Schaltfläche geklickt wird.

Wir könnten dies, wie bei Java-Swing üblich, mit einem Listener machen, den wir dem Button-Objekt zuweisen. Java-Android macht es uns an dieser Stelle sogar noch einfacher. Wir können in der XML-Ansicht der Layout-Datei dem Button-Objekt mit dem Befehl `android:onClick="methodenname"` eine Methode zuweisen, die dann immer automatisch aufgerufen wird.

Hier ist ganz wichtig, dass Methoden, die von einem View-Objekt aufgerufen werden sollen, folgender Methodensyntax genügen:

Code 4.5: Methodensyntax für Methoden, die von View-Objekten aufgerufen werden

```
1 public void methodenname(View view) {
2     ...
3 }
```

D.h. alle Methoden, die von View-Objekten aufgerufen werden, benötigen als Übergabeparameter ein Objekt der Klasse View.

Bei uns würde das konkret wie folgt aussehen. Wir schreiben eine Methode `changeBackgroundColor` in der Activity-Klasse und möchten gerne, dass diese immer aufgerufen wird, falls auf unser Button-Objekt geklickt wird.
Dazu geben wir der Methode zunächst folgende Syntax:

Code 4.6: Methodensyntax für Methoden, die von View-Objekten aufgerufen werden

```
1 public void changeBackgroundColor(View view) {
2     ...
3 }
```

In der XML-Ansicht weisen wir nun diese Methode dem Button-Objekt zu:

```
<Button
    android:id="@+id/changeColorButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="58dp"
    android:onClick="changeBackgroundColor"
    android:text="@string/changecolorbutton_text" />
```

4.5 Zugriff auf View-Objekte aus Activity-Klassen

In der gerade erstellten Methode `changeBackgroundColor` möchten wir nun gerne die Hintergrundfarbe ändern. Dazu müssen wir auf das `RelativeLayout` der Layout-Datei zurückgreifen, da dieses den Hintergrund unserer App bildet. Nun stellt sich die Frage, wie wir aus einer Activity-Klasse auf View-Objekte der Layout-Datei zugreifen können.

Wenn wir etwa bei Klicken auf unseren Button dessen Beschriftung ändern möchten, müssen wir in der Methode

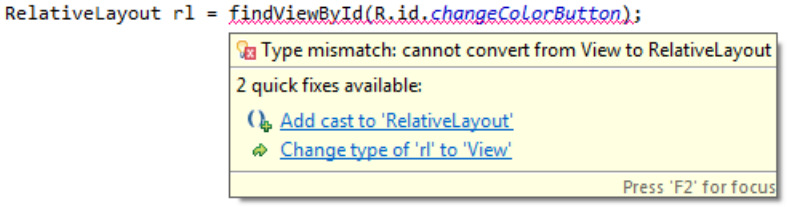
auch irgendwie auf das Button-Objekt zugreifen können.

An dieser Stelle kommen die IDs ins Spiel. Betrachten wir die XML-Ansicht genauer, sehen wir, dass auch das `RelativeLayout` eine `ID` hat:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.einfuehrungsaufgabe.MainActivity"
    tools:ignore="MergeRootFrame" >
```

Mit dem Befehl `findViewById(R.id.idname)` können wir aus einer Activity-Klasse auf das View-Objekt mit der `ID` *idname* zugreifen.

Möchten wir nun das mit obigen Befehl gefundene View-Objekt in einer Variable vom Datentyp `RelativeLayout` speichern, gibt Eclipse eine Fehlermeldung aus:



Mit dem Befehl `findViewById(R.id.idname)` können wir erst einmal jedes x-beliebige View-Objekt „finden“. Möchten wir dieses in einer Variable eines bestimmten Datentyps speichern, müssen wir zusichern, dass das gefundene Objekt auch wirklich von diesem Datentyp ist. Dies geschieht mit einem `cast`. Damit garantieren wir der Variable, dass ihr auch wirklich ein Objekt ihres Datentyps zugewiesen wird. Ist dies zur Laufzeit nicht der Fall, wird während der Kompilierung ein Fehler geworfen.

Allgemein sieht das Speichern eines View-Objektes mit der `ID` *idname* in einer Variable *variablenname* vom Datentyp *Datentyp* also wie folgt aus:

Code 4.7: Zugriff aus View-Objekte aus einer Activity-Klasse

```
1 Datentyp variablenname = (Datentyp) findViewById(R.id.idname);
```

Für unseren konkreten Fall mit dem `RelativeLayout` bedeutet dies somit:

Code 4.8: Zugriff aus View-Objekte aus einer Activity-Klasse

```
1 RelativeLayout rl = (RelativeLayout) findViewById(R.id.container);
```

Unsere Methode zum Ändern der Hintergrundfarbe des `RelativeLayouts` sieht somit folgendermaßen aus:

Code 4.9: Methode zum Aendern der Hintergrundfarbe

```
1 /**
2  * Methode zum Aendern der Hintergrundfarbe
3  * @param view
4  */
5 public void changeBackgroundColor(View view) {
6     RelativeLayout rl = (RelativeLayout) findViewById(R.id.changeColorButton);
7     rl.setBackgroundColor(Color.YELLOW);
8 }
```

4.6 Übungsaufgaben

Wir schließen das Kapitel mit zwei Übungsaufgaben. Sie sollten diese Aufgaben ohne größere Probleme bearbeiten können, bevor Sie sich den folgenden Kapiteln zuwenden.

 **Übung 1** 

Bisher können wir die Hintergrundfarbe nur ein einziges Mal von weiß auf gelb ändern. Ändern Sie Ihre App so ab, dass

(a) die Hintergrundfarbe abwechselnd auf weiß bzw. gelb gesetzt wird.

(b) neben dem Wechseln der Hintergrundfarbe auch die Beschriftung des Button-Objekts zwischen *Hintergrundfarbe auf gelb setzen* und *Hintergrundfarbe auf weiß setzen* wechselt.

(c) die Hintergrundfarbe auf eine zufällige Farbe einer bestimmten Vorauswahl gesetzt wird.

 **Übung 2** 

Erstellen Sie zusätzlich zu dem Button-Objekt ein Objekt der Klasse `EditText`. Dieses befindet sich in der Rubrik `Text Fields`. Bei Klicken auf die Schaltfläche *Hintergrundfarbe ändern* soll nun der in das Textfeld eingegebene Text als neue Beschriftung der Schaltfläche festgelegt werden. Wurde kein Text eingegeben, soll nichts passieren. Bei jeder Änderung soll sich weiter auch noch die Hintergrundfarbe ändern:

Die Lösungen zu Übung 1 finden Sie auf Seite 116 und die Lösungen zu Übung 2 auf Seite 117.

Kapitel 5

Englisch-Vokabel-App

Überblick:

Dieses Kapitel dient dazu, Ihnen einen ersten Einblick in die App-Programmierung mit Java Android zu ermöglichen. Da wir in diesem Kapitel auf das in Kapitel 4 Gelernte zurückgreifen und darauf aufbauen werden, wird dringend empfohlen, sich zunächst mit Kapitel 4 auseinanderzusetzen und die Übungsaufgaben am Ende ohne Probleme bearbeiten zu können. Sie werden bei dieser Aufgabe neben dem Zusammenspiel und der Interaktion verschiedener Activity-Klassen auch die Anbindung an eine appinterne Datenbank kennenlernen.

5.1 Aufgabenstellung

Wir möchten eine App programmieren, mit der wir unsere gelernten Vokabeln (hier: Englisch) abspeichern und anschließend abfragen können. Dabei sollen wir auswählen können, ob wir die Vokabeln von Deutsch nach Englisch, von Englisch nach Deutsch oder in zufälliger Richtung abfragen möchten. Neben dem Abspeichern und Abfragen der Vokabeln möchten wir auch die Möglichkeit haben, gespeicherte Vokabeln nachträglich noch zu editieren, um etwaige Tippfehler korrigieren zu können, und auch Vokabeln zu löschen.

5.2 Projekt anlegen

Zunächst müssen wir ein `Android-Applikation-Projekt` anlegen. Wir gehen hier genauso vor, wie in Kapitel 4 in Abschnitt 4.1. Als App-Namen verwenden wir *Englisch-Vokabeln*[1] und als App-Symbol ein geeignetes Bild:

5.3 MainActivity und Layout

Zunächst ändern wir die Klasse `MainActivity.java`[2] leicht ab, da wir einige generierte Einstellungen nicht benötigen:

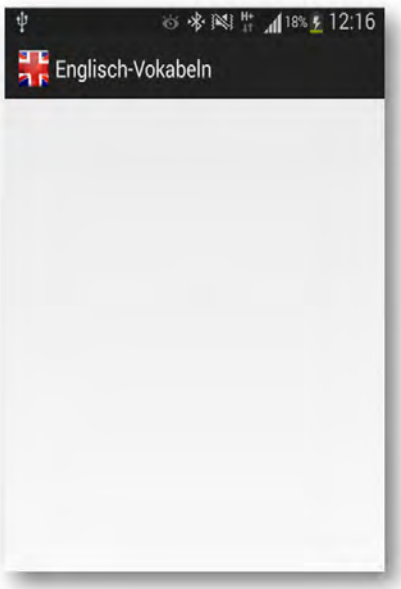
Code 5.1: MainActivity.java

```
1 package com.example.englisch_vokabeln;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5 import android.view.Menu;
6
7 public class MainActivity extends Activity {
8
9
```

[1] Beachten Sie auch, dass Sie als `Minimum Required SDK API 16` festlegen.
[2] `Englisch-Vokabeln` → `src` → `com.example.englisch_vokabeln`

```
10    @Override
11    protected void onCreate(Bundle savedInstanceState) {
12      super.onCreate(savedInstanceState);
13      setContentView(R.layout.activity_main);
14    }
15  }
```

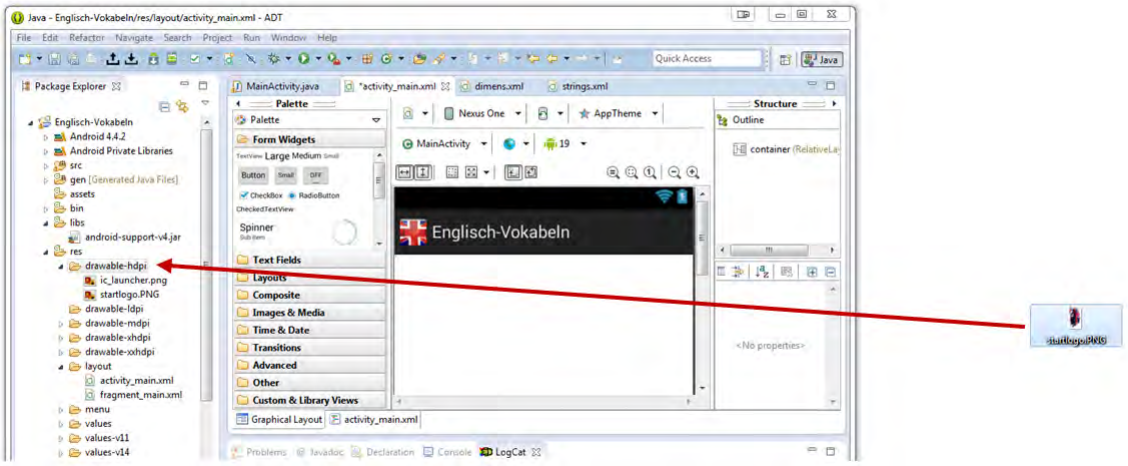
Wir können an dieser Stelle unsere App einfach mal starten: dazu verbinden wir unser Mobile Device mit dem Computer, wählen in Eclipse die Datei `MainActivity.java` aus und drücken auf das  - Symbol. Wir wählen `Run as` `Android Application`.
Wurde unser Mobile Device vom Computer als ADB-Interface erkannt, wird es in der Liste angezeigt (vgl. hier Seite 19). Wir wählen es aus und drücken auf `Ok`. Nach kurzer Wartezeit wird die Applikation auf dem Handy angezeigt:



Wir möchten bei Aufruf der App einen Startbildschirm mit einem Foto von England anzeigen (s. Abbildung auf Seite 26). Wenn man auf dieses Foto klickt, soll ein Menü angezeigt werden, in welchem man auswählen kann, ob man neue Vokabeln hinzufügen oder gespeicherte Vokabeln abfragen möchte.

Wir weisen dazu der Layout-Datei `activity_main.xml` ein `RelativeLayout` zu und erstellen ein Button-Objekt, das wir in die Mitte des virtuellen Bildschirms ziehen. Dem Button-Objekt möchten wir nun ein Hintergrundbild zuweisen. Dazu müssen wir das Bild zunächst im Ordner `drawable-hdpi`[3] abspeichern. Dies geht ganz einfach mit Drag & Drop:

[3] dieser Ordner befindet sich im Ordner `res`



In der XML-Ansicht können wir mit dem Befehl `android:background="..."` dem Button-Objekt eine Bild-Datei als Hintergrund zuweisen. Als Button-Beschriftung wählen wir den leeren String:

Code 5.2: XML-Ansicht von `activity_main.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:background="@drawable/startlogo"
        android:text="" />

</RelativeLayout>
```

5.4 Informationen und Nachrichten anzeigen: Toast

An manchen Stellen möchte man den App Nutzer Informationen über kleine Nachrichten zukommen lassen. Dies geht in Java Android über sog. Toasts. Beispielsweise soll bei Start der Applikation eine Nachricht erscheinen, dass man auf das Bild klicken soll. Dazu legt man in der `onCreate`-Methode der `MainActivity`-Klasse einen Toast an und zeigt diesen an:

Code 5.3: Toasts zum Ausgeben von Informationen und Benachrichtigungen

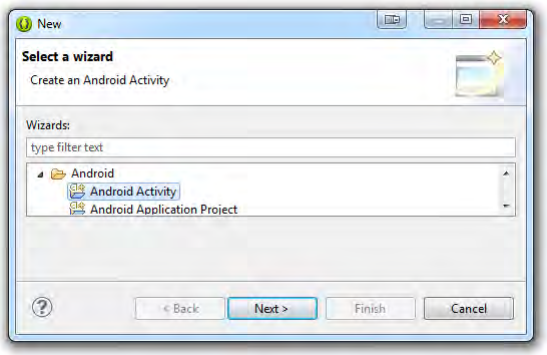
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  // Toast, um App Nutzer z.B. Informationen mitteilen zu koennen
  Toast t = Toast.makeText(this, "Auf das Bild klicken, um App zu starten", Toast.LENGTH_LONG);
  t.show();
}
```

Falls man auf das Toast.Objekt im Folgenden nicht mehr zugreifen muss, was fast immer der Fall sein wird, kann man sich das Erzeugen des Objektes auch sparen:

```
Toast.makeText(this, "Auf das Bild klicken, um App zu starten", Toast.LENGTH_LONG).show();
```

5.5 Neue Activity-Klasse erstellen

Durch Klicken auf den Button soll uns ein Auswahlmenü angezeigt werden. Dafür erstellen wir zunächst eine neue Activity-Klasse - wir nennen diese `SelectionActivity.java`. Dazu machen wir einen Rechtsklick auf das Package `com.example.englisch_vokabeln` im `src`-Ordner und klicken auf `New` → `Other` → `Android` → `Android Activity`:



Wir löschen auch in der `SelectionActivity`-Klasse wieder alle überflüssigen Methoden:

Code 5.4: SelectionActivity.java

```
1  package com.example.englisch_vokabeln;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class SelectionActivity extends Activity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_selection);
12     }
13 }
```

Neben der Activity-Klasse wurde auch gleichzeitig die zugehörige Layout-Datei `activity_selection.xml` im Ordner `layout` erstellt.

5.6 Wechseln zwischen zwei Activity-Klassen

Als nächstes müssen wir festlegen, dass diese Activity-Klasse aufgerufen wird, wenn wir auf das Bild (Button) drücken. Dazu fügen wir in der Klasse `MainActivity` folgende Methode hinzu:

Code 5.5: Methode zum Aufrufen der Activity `SelectionActivity`

```
1  public void startSelectionActivity(View view) {
2    Intent myIntent = new Intent(this, SelectionActivity.class);
3    startActivity(myIntent);
4  }
```

Mit diesen Befehlen wird die in `myIntent` festgelegte Activity-Klasse aufgerufen. Wir müssen diese Methode nun noch unserem Button-Objekt zuweisen[4]. Wie das geht, haben Sie in Kapitel 4 in Abschnitt 4.4 gesehen.

[4] Aus diesem Grund haben wir auch gleich den Aufrufparameter `View` verwendet.

Testen Sie die Applikation, indem Sie diese auf Ihrem Mobile Device ausführen und starten Sie durch Klicken auf das Bild die neue Activity-Klasse `SelectionActivity`.

5.7 Die Klasse SelectionActivity

In diesem Abschnitt kümmern wir uns um das Layout und die Funktionalität der Klasse `SelectionActivity`. Wir haben im vorangehenden Abschnitt bereits die zugehörige Layout-Datei `activity_selection.xml` kurz angesprochen.
In dieser Layout-Datei verwenden wir erneut das `RelativeLayout` und fügen ein TextView-Objekt und zwei Button-Objekte hinzu. Verwenden Sie für die Button-Objekte die `IDs` *addbutton* und *voctestbutton*, oder andere aussagekräftige Namen.
Für die Button-Objekte haben wir uns mit einem Textverarbeitungsprogramm ein Hintergrundbild erstellt:

Ihr virtueller Bildschirm und die zugehörige XML-Ansicht sollten etwa wie folgt aussehen:

Code 5.6: XML-Ansicht von `activity_selection.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/RelativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/white"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="19dp"
```

```
16          android:layout_marginTop="46dp"
17          android:text="@string/selection_text" />
18
19      <Button
20          android:id="@+id/addbutton"
21          android:layout_width="match_parent"
22          android:layout_height="75dp"
23          android:layout_alignParentLeft="true"
24          android:layout_below="@+id/textView1"
25          android:layout_marginTop="40dp"
26          android:background="@drawable/buttonbg"
27          android:text="@string/addbutton_text"
28          android:textSize="15sp" />
29
30      <Button
31          android:id="@+id/voctestbutton"
32          android:layout_width="match_parent"
33          android:layout_height="75dp"
34          android:layout_below="@+id/addbutton"
35          android:background="@drawable/buttonbg"
36          android:text="@string/voctestbutton_text"
37          android:textSize="15sp" />
38
39  </RelativeLayout>
```

 **Aufgabe 1** 

Über die erste Schaltfläche möchten wir eine Activity-Klasse aufrufen, in der wir neue Vokabeln hinzufügen können. Über die zweite Schaltfläche möchten wir eine Activity aufrufen, in der wir zuvor eingegebene Vokabeln abfragen können.

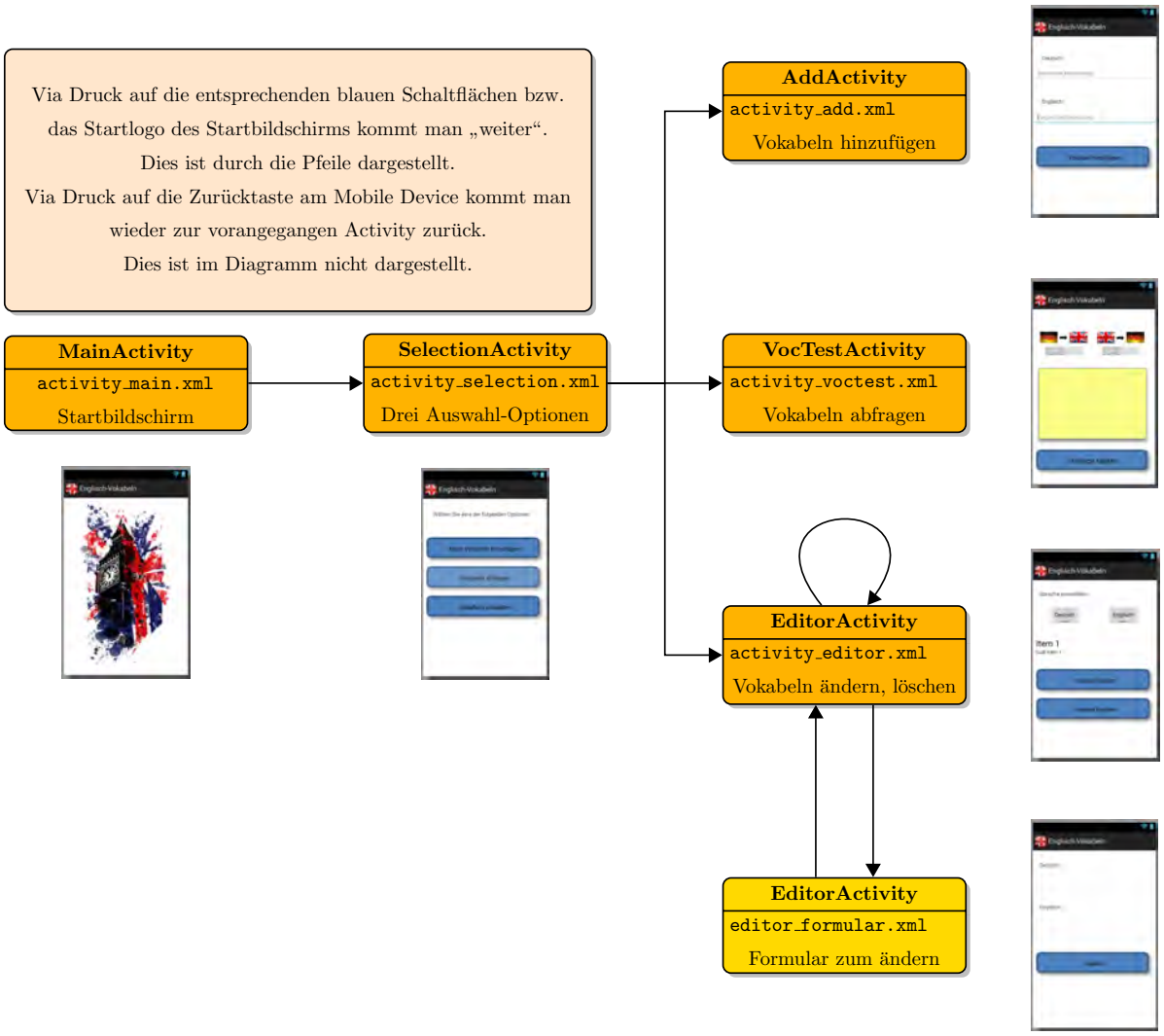
(a) Erstellen Sie dazu geeignete Activity-Klassen mit den Namen `AddActivity` und `VocTestActivity`.

(b) Legen Sie fest, dass die jeweilige Activity-Klasse durch Klicken auf die zugehörige Schaltfläche in der `SelectionActivity` aufgerufen wird.

Testen Sie die Applikation auf Ihrem Mobile Device und navigieren Sie zwischen den einzelnen Activities hin und her. Mit der Zurück-Schaltfläche Ihres Mobile Device können Sie zur vorherigen Activity zurückspringen.

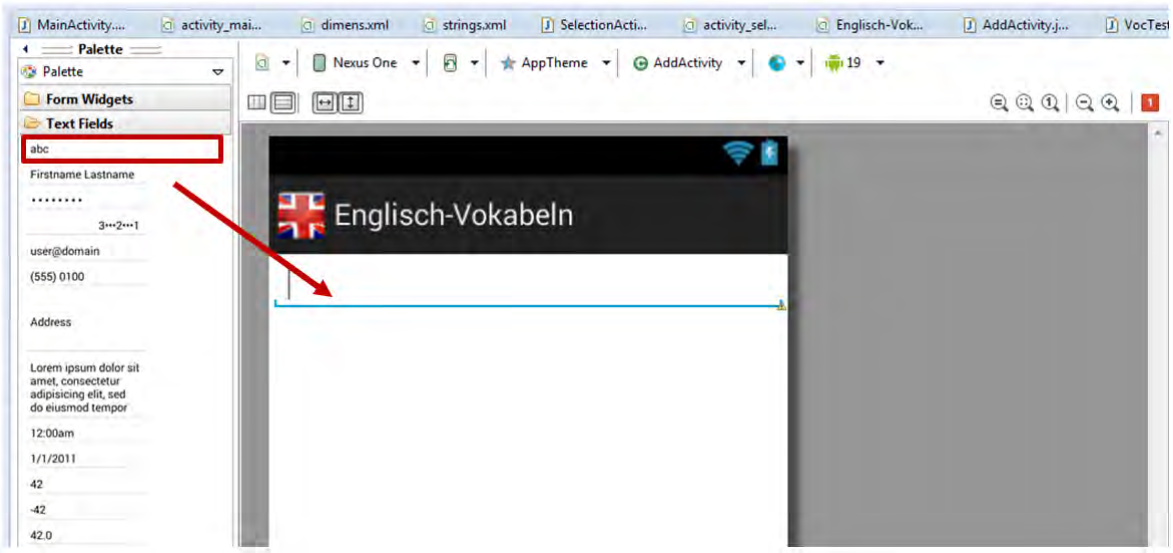
5.8 Zustandsdiagramm zur Benutzerführung

Wir haben nun bereits mehrere Activity-Klassen erstellt. In unten abgebildeter Grafik sehen Sie einen Überblick über alle verwendeten Activity-Klassen in unserer Englisch-App. Die Klasse `EditorActivity` erstellen wir erst in Abschnitt 5.14.1, haben diese hier aber trotzdem schon mal aufgeführt:

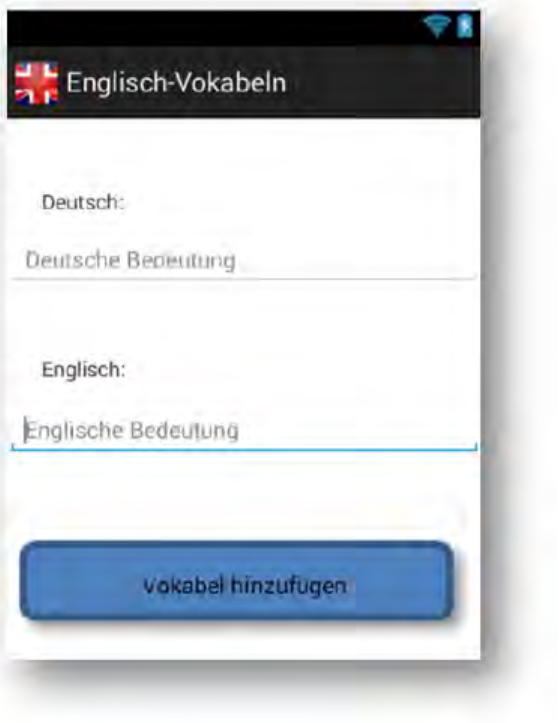


5.9 Layout der AddActivity-Klasse

Die AddActivity-Klasse soll eine Art Formular anzeigen, in dem wir die englische und deutsche Bedeutung einer Vokabel eintragen können und mittels einer Schaltfläche abspeichern können.
Dazu verwenden wir u.a. zwei Objekte der Klasse EditText (aus der Kategorie `Text Fields`).:



Das Formular könnte etwa wie folgt aussehen:



Wir haben dem EditText-Objekt für die deutsche Bedeutung die ID `editGerman`, dem EditText-Objekt für die englische Bedeutung die ID `editEnglish` und dem Button die ID `vocaddButton` zugewiesen.

5.10 Datenbankanbindung - die Klasse DataHandler

Um Vokabeln abspeichern zu können, benötigen wir eine applikationsinterne Datenbank. Wir nennen diese `vocDB`.

Für das Erzeugen und den späteren Zugriff auf diese Datenbank erstellen wir uns eine Klasse `DataHandler`.

Unsere Datenbank soll die Tabelle `vocTable` mit den beiden Spalten `deutsch` und `englisch` haben. Dazu erstellen wir in unserer Klasse geeignete String-Konstanten:

Code 5.7: Die Klasse `DataHandler.java`

```
1  package com.example.englisch_vokabeln;
2
3  import android.content.Context;
4
5  public class DataHandler {
6
7    // String-Konstanten fuer Datenbank
8    public static final String DATABASE_NAME = "vocDB";
9    public static final int DATABASE_VERSION = 1;
10
11   // String-Konstanten fuer Tabellen- und Spaltennamen
12   public static final String TABLE_NAME = "vocTable";
13   public static final String DEUTSCH = "deutsch";
14   public static final String ENGLISCH = "englisch";
15
16   public DataHandler(Context ctx) {
17
18   }
19 }
```

5.10.1 Die innere Klasse `DataBaseHelper`

In der Klasse `DataHandler` erstellen wir eine private innere Klasse `DataBaseHelper`, die von der Klasse `SQLiteOpenHelper` erbt:

Code 5.8: Die innere Klasse `DataBaseHelper.java`

```
1  private static class DataBaseHelper extends SQLiteOpenHelper {
2
3    public DataBaseHelper(Context ctx) {
4      super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
5    }
6
7    @Override
8    public void onCreate(SQLiteDatabase db) {
9      // TODO Auto-generated method stub
10   }
11
12   @Override
13   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
14     // TODO Auto-generated method stub
15   }
16 }
```

In der `onCreate`-Methode der inneren Klasse `DataBaseHelper` wollen wir die Tabelle `vocTable` mit den Spalten `deutsch` und `englisch` erstellen, falls diese noch nicht existiert.

Aufgabe 2

Erstellen Sie einen SQL-Befehl, um oben beschriebene Tabelle `vocTable` mit den Spalten `deutsch` und `englisch` zu erstellen.

Hinweis: SQLite verwendet für Text anstatt des Datentyps VARCHAR den Datentyp TEXT.

Diesen Befehl können wir mit der Methode `execSQL(..)` direkt im Java-Quellcode ausführen:

Code 5.9: `onCreate(..)`-Methode der Klasse `DataBaseHelper`

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table if not exists vocTable(deutsch text not null, englisch text not null);");
}
```

5.10.2 Methoden der Klasse `DataHandler`

Wir erzeugen in der Klasse `DataHandler` im Konstruktor ein neues Objekt der Klasse `DataBaseHelper`. Zudem legen wir Variablen für `Context` und `SQLiteDataBase` an:

Code 5.10: Konstruktor der Klasse `DataHandler.java`

```
private DataBaseHelper dbHelper;
private Context ctx;
private SQLiteDatabase db;

/*
 * Konstruktor der Klasse DataHandler
 */
public DataHandler(Context ctx) {
    this.ctx = ctx;
    dbHelper = new DataBaseHelper(ctx);
}
```

Um später lesend oder schreibend auf unsere Datenbank zugreifen zu können, benötigen wir noch folgende drei Methoden:

Code 5.11: Methoden für Lese- bzw. Schreibzugriff auf die Datenbank

```
/**
 * Speichert in der Variable db unsere Datenbank mit Schreib-Rechten
 */
public void openWrite() {
  db = dbHelper.getWritableDatabase();
```

```
6  }
7
8  /**
9   * Speichert in der Variable db unsere Datenbank mit Lese-Rechten
10  */
11 public void openRead() {
12   db = dbHelper.getReadableDatabase();
13 }
14
15 /**
16  * Nach Ausfuehrung von Lese- oder Schreibzugriffen werden die Zugriffsrechte auf die Datenbank
17  * wieder zurueckgesetzt
18  */
19 public void close() {
20   dbHelper.close();
21 }
```

Wir brauchen noch eine Methode, um einen neuen Eintrag in unsere Tabelle `vocTable` in der Datenbank machen zu können. Dazu benötigen wir zuerst wieder einen SQL-Befehl, um ein neues Vokabelpaar in die Tabelle einzufügen.

Aufgabe 3

(a) Erstellen Sie einen SQL-Befehl, um ein neues Vokabelpaar in die Tabelle `vocTable` mit den Spalten `deutsch` und `englisch` einfügen zu können.

(b) Erstellen Sie anschließend eine Methode `addEntry`, die als Übergabeparameter die deutsche und englische Bedeutung erhält und diese in die Tabelle `vocTable` einfügt.

Hinweis: Den SQL-Befehl können Sie mit `db.execsql(...)` direkt im Java-Code ausführen.

5.11 Vokabeln abspeichern

5.11.1 Die Methode `addButtonClicked`

Im vorherigen Abschnitt haben wir die Klasse DataHandler, über die wir die applikationsinterne SQLite-Datenbank erstellen und auf diese zugreifen können. Wir haben bereits eine Methode implementiert, der wir die deutsche und englische Bedeutung der Vokabel als Parameter übergeben und die diese anschließend in die Tabelle `vocTable` der Datenbank `vocDB` einträgt.

In der Klasse AddActivity wollen wir neu eingegebenen Vokabeln abspeichern. Somit erstellen wir hier ein Objekt der Klasse DataHandler und initialisieren es bei Aufruf der Activity-Klasse:

Code 5.12: Initialisieren des DataHandler-Objektes in der `onCreate`-Methode der Klasse `AddActivity`

```
1 private DataHandler dbHandler;
2
3 @Override
4 protected void onCreate(Bundle savedInstanceState) {
5   ...
6   //Initialisieren des DataHandler-Objekts
7   dbHandler = new DataHandler(this);
8 }
```

Nun benötigen wir noch eine Methode, die aufgerufen wird, wenn auf die Schaltfläche *Vokabel hinzufügen* geklickt wird. Wir wollen die Vokabel nur abspeichern, falls sowohl eine deutsche als auch eine englische Bedeutung eingegeben wurde. Ansonsten soll der Benutzer benachrichtigt werden, dass die Eingabe ungültig ist.

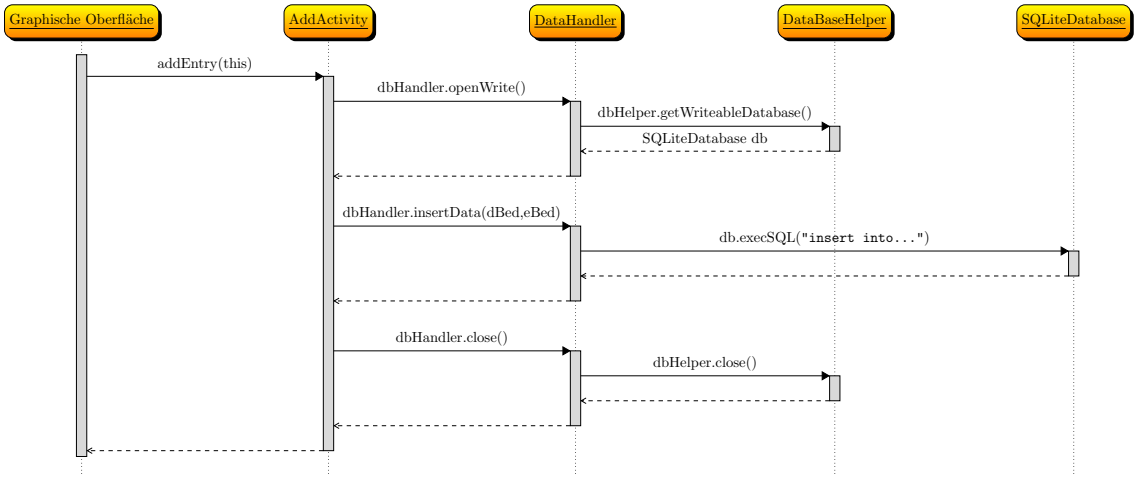
Aufgabe 4

(a) Erstellen Sie eine Methode `addButtonClicked` in der Klasse `AddActivity` und sorgen Sie dafür, dass diese bei Klicken auf die Schaltfläche *Vokabel hinzufügen* auch aufgerufen wird.

(b) Erstellen Sie in dieser Methode eine Kontrollstruktur, mit der Sie überprüfen, ob auch wirklich sowohl eine deutsche als auch eine englische Bedeutung eingegeben wurde. Ist dies nicht der Fall geben Sie eine geeignete Benachrichtigung aus.

(c) Wurde sowohl eine deutsche als auch eine englische Bedeutung eingegeben, sollen diese in der Datenbank gespeichert werden. Nutzen Sie hierzu die entsprechende Methode der Klasse `DataHandler`.

Hinweis: Beachten Sie, dass Sie zunächst Schreibrechte anfordern müssen. Nach dem Einfügen der Vokabel soll die Berechtigung wieder zurückgesetzt werden.
Verwenden Sie hierzu die Befehle `db.openWrite()` und `db.close()`. Sie können sich an unten abgebildeten Sequenzdiagramm orientieren:



Lassen Sie sich nicht von dem Parameter `this` der `addEntry`-Methode verwirren. Das View-Objekt (hier die Schaltfläche) gibt sich beim Methodenaufruf selbst zurück. Aus diesem Grund auch der Aufrufparameter von der Oberklasse `View`.

Wir können nun unsere ersten Vokabeln abspeichern. Wir speichern zunächst die Vokabeln *(eins,one), (zwei,two), (drei,three)* ab, um später das Abfragen von Vokabeln testen zu können.

5.11.2 Anzahl der Einträge

Wir möchten bei erfolgreichem Hinzufügen einer neuen Vokabel eine Benachrichtigung ausgeben, wie viele Vokabeln sich in der Datenbank befinden.
Dazu brauchen wir zunächst eine Methode in der Klasse `DataHandler`, die uns die Anzahl der Einträge der Tabelle `vocTable` zurückgibt.

Aufgabe 5

Erstellen Sie eine SQL-Anfrage, die die Anzahl der Einträge der Tabelle `vocTable` liefert.

SQL-Befehle, die Anfragen an die Datenbank stellen und somit einen Rückgabewert (z.B. ein bestimmtes Vokabelpaar, Anzahl der Einträge, etc.) besitzen, führen wir mit dem Befehl `db.rawQuery( .. )`. Dieser Befehl liefert als Rückgabewert ein Objekt der Klasse `Cursor`. Für unseren Fall würde das wie folgt aussehen:

```
1 Cursor c = db.rawQuery("select count(*) from vocTable;", null);
```

Möglicherweise hören Sie an dieser Stelle zum ersten Mal von einem `Cursor-Objekt`. Im nächsten Unterabschnitt finden Sie eine Beispielanfrage an die Datenbank, um Ihnen die Funktionsweise des Cursor-Objekts näher zu bringen.

5.11.3 Funktionsweise eines Cursors anhand eines Beispiels

Nehmen wir an, die Tabelle `vocTable` hätte folgende Einträge

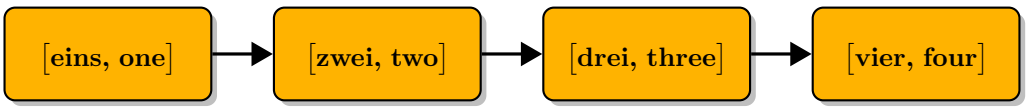
eins	one
zwei	two
drei	three
vier	four

Tabelle 5.1: `vocTable`.

und wir möchten mit einer SQL-Anfrage alle Einträge der Tabelle `vocTable` auslesen. In Java-Code hätte diese Anfrage dann folgenden Befehl:

```
1 Cursor c = db.rawQuery("select deutsch,englisch from vocTable;", null);
```

Das Cursor-Objekt kann man sich grafisch folgendermaßen vorstellen:



D.h. für jede Zeile wird ein Cursor-Eintrag erstellt (orange Box). In den orangen Boxen finden sich dann so viele Einträge, wie eine Zeile Spalten hat, in unserem Fall zwei.

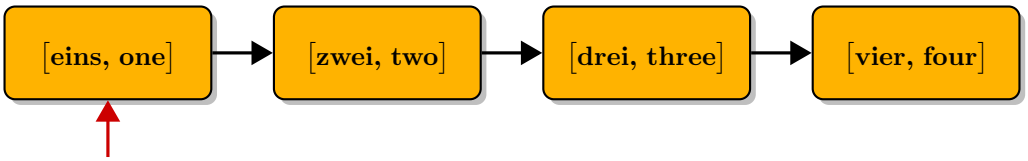
Beachte: Angenommen wir hätten in unserer Tabelle noch eine dritte Spalte mit der spanischen Bedeutung und wir würden obige SQL-Anfrage stellen. Dann würde das Cursor-Objekt immer noch genauso aussehen, d.h. die Anzahl der Einträge in einem Cursor-Element richtet sich nicht nach der Anzahl der Spalten der Tabelle, sondern nach den Anzahl der Spalten, die bei der SQL-Anfrage abgefragt werden.

Möchte man nun auf die Elemente des Cursors schrittweise durchlaufen, setzt man zunächst einen Zeiger auf das erste Element[5]. Dies geht wie folgt:

```
if (c.moveToFirst()) {
    // Zeiger auf erstes Element setzen,
    // falls möglich
} else {
    // Fehlermeldung
}
```

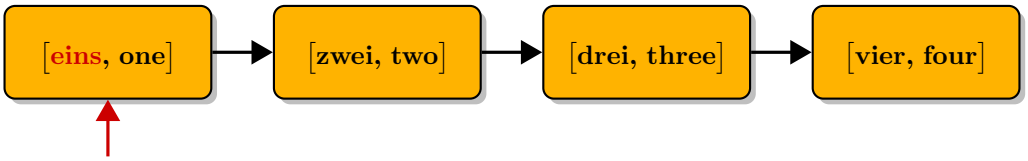
[5]Die Methode `moveToFirst()` hat als Rückgabewert einen `boolean`. Sie liefert `true`, falls es möglich ist, den Zeiger auf das erste Element zu setzen und `false` falls nicht, also falls der Cursor leer ist

Grafisch könnte man diesen Schritt wie folgt deuten:



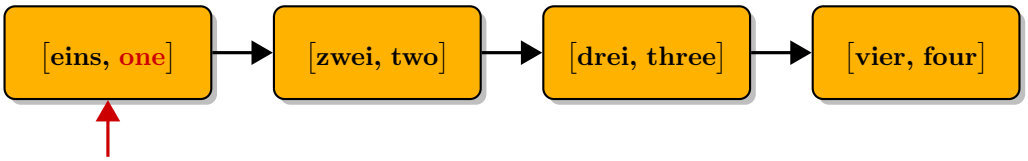
Auf die einzelnen Einträge eines Cursor-Elements kann mit dem Befehl `c.getString(i)` zugreifen. i steht hier für den i-ten Eintrag eines Cursor-Elements[6].
Auf den ersten Eintrag des ersten Cursor-Elements kann man also wie folgt zugreifen:

```
if (c.moveToFirst()) {
    String s1 = c.getString(0);
    String s2 = c.getString(1);
} else {
    // Fehlermeldung
}
```



Und auf den zweiten Eintrag des ersten Cursor-Elements folglich mit:

```
if (c.moveToFirst()) {
    String s1 = c.getString(0);
    String s2 = c.getString(1);
} else {
    // Fehlermeldung
}
```

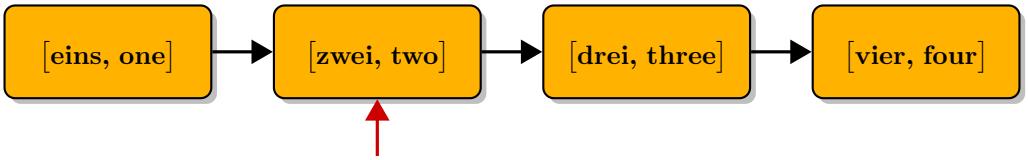


Nun haben wir beide Einträge des ersten Cursor-Elements ausgelesen und möchten zum nächsten Cursor-Element springen. Dies geht mit dem Befehl `c.moveToNext()`. Dieser Befehl liefert `true`, falls das Springen zum nächsten Element möglich ist und `false`, falls nicht. Wird `false` zurückgegeben heißt das, dass man aktuell auf das letzte Element im Cursor zugreift.
Wir können also per `while`-Wiederholung schrittweise alle Elemente des Cursors durchlaufen und jeweils auf die zwei Einträge zugreifen:

[6] Beachten Sie, dass wie so oft bei 0 angefangen wird zu zählen. Unsere Cursor-Element haben also einen nullten und einen ersten Eintrag

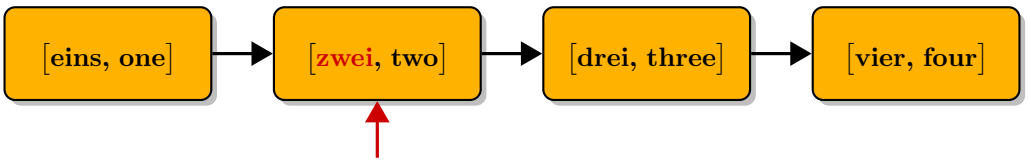
Springen zum nächsten Cursor-Element:

```
if (c.moveToFirst()) {
    String s1 = c.getString(0);
    String s2 = c.getString(1);

    while(c.moveToNext()) {
        s1 = c.getString(0);
        s2 = c.getString(1);
    }
} else {
    // Fehlermeldung
}
```



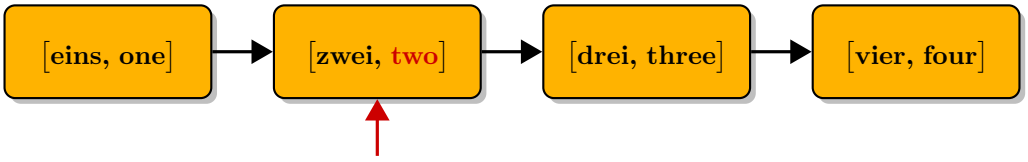
Zugriff auf ersten Eintrag:

```
if (c.moveToFirst()) {
    String s1 = c.getString(0);
    String s2 = c.getString(1);

    while(c.moveToNext()) {
        s1 = c.getString(0);
        s2 = c.getString(1);
    }
} else {
    // Fehlermeldung
}
```



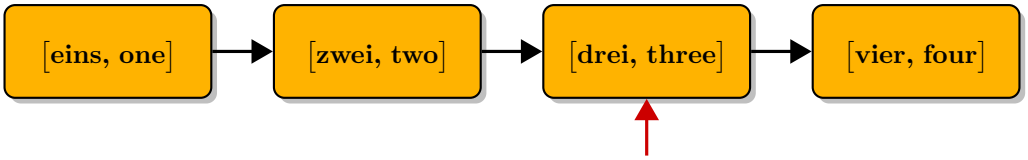
Zugriff auf zweiten Eintrag:

```
if (c.moveToFirst()) {
    String s1 = c.getString(0);
    String s2 = c.getString(1);

    while(c.moveToNext()) {
        s1 = c.getString(0);
        s2 = c.getString(1);
    }
} else {
    // Fehlermeldung
}
```



Springen zum nächsten Cursor-Element:

```
if (c.moveToFirst()) {
    String s1 = c.getString(0);
    String s2 = c.getString(1);

    while(c.moveToNext()) {
        s1 = c.getString(0);
        s2 = c.getString(1);
    }
} else {
    // Fehlermeldung
}
```



usw ... .

5.11.4 Fortsetzung: Anzahl der Einträge

Aufgabe 6

Erstellen Sie eine Methode `tableSize()` in der Klasse `DataHandler`, die die Anzahl der Einträge der Tabelle `vocTable` zurückgibt. Machen Sie sich Gedanken,

(a) welchen Rückgabetyp Ihre Methode haben soll.

(b) wie Sie die Anzahl der Einträge auslesen können.

(c) wie viele Elemente und wie viele Einträge in einem Element das Cursor-Objekt Ihrer Anfrage hat. Überlegen Sie sich hier weiter, wie Sie die Anzahl der Einträge aus dem Cursor-Objekt auslesen können.

Diese Methode können wir nun nutzen und damit die Methode `addButtonClicked` erweitern:

```
1  public void addButtonClicked(View view) {
2      ...
3      // War eines oder beide Eingabefelder leer, wird eine Fehlermeldung angezeigt
4      if(deutscheBedeutung.equals("") || englischeBedeutung.equals("")) {
5          ...
6      } else {
7          ...
8          // Anzahl der Eintraege abfragen
9          dbHandler.openRead();
10         int size = dbHandler.tableSize();
11         dbHandler.close();
12
13         Toast.makeText(this, "Vokabel gespeichert - Anzahl der Eintraege: " + size, Toast.LENGTH_LONG).show
               ();
```

```
14      }
15  }
```

Wir testen die abgeänderte Methode und fügen noch die Einträge `(vier,four)` und `(fünf,five)` hinzu. Dabei sollte folgende Benachrichtigung angezeigt werden:

5.12 Layout der VocTestActivity-Klasse

In diesem Abschnitt beschäftigen wir uns mit dem Aufbau der VocTestActivity-Klasse.

5.12.1 Abfragerichtung festlegen

Wir sollen auswählen können, ob wir die Vokabeln von Deutsch nach Englisch, von Englisch nach Deutsch oder zufällig abfragen möchten.
Zur Visualisierung haben wir uns zwei Bilder erstellt:

Um die Bilder anzeigen zu können, erstellen wir zwei TextView-Objekte ohne Text und weisen ihnen je eines der zwei Bilder zu. Um nun eine oder beide Richtungen auswählen zu können, erstellen wir uns zwei Switch-Objekte[7] und platzieren diese unter den Bildern[8]. Wir haben den Switch-Objekten die IDs `deutschenglischswitch` und

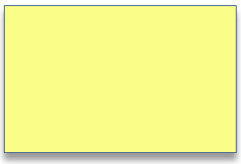
[7] Diese findet man auch in der Kategorie `Form Widgets`.
[8] Wir haben ein `RelativeLayout` verwendet.

`englischdeutschswitch` zugewiesen.

Das Layout der VocTestActivity-Klasse sollte bisher ungefähr wie folgt aussehen:

5.12.2 Vokabelkarten

Wir möchten die Vokabeln auf einer Karteikarte anzeigen. Dazu haben wir uns mit einem Zeichenprogramm eine Karteikarte erstellt und als `PNG`-Datei in unsere Android-Projekt importiert:



Wir erstellen ein Button-Objekt ohne Text und weisen diesem die Bild-Datei als Hintergrund zu und geben diesem die ID `vocKarte`. Anschließend erstellen wir noch ein Button-Objekt mit dem Text *Abfrage starten* und der ID `vocButton` (wir bezeichnen dieses im Folgenden als *Anzeigen*-Schaltfläche).

Das Layout der VocTestActivity-Klasse sollte nun ungefähr wie folgt aussehen:

5.13 Vokabeln abfragen

Nachdem wir im vorherigen Abschnitt das Layout der VocTestActivity-Klasse festgelegt haben, beschäftigen wir uns nun mit ihrer Funktionalität.

Wir müssen zwei Fälle unterscheiden, wenn auf das Button-Objekt geklickt wird:

- am Anfang oder falls aktuell eine Lösung[9] angezeigt wird, soll eine neue Vokabel angezeigt werden
- wird aktuell eine Vokabel angezeigt, soll bei Klicken auf das Button-Objekt die zugehörige Lösung angezeigt werden

Um diese beiden Fälle unterscheiden zu können, führen wir eine boolsche Variable `showVoc` ein, die `true` ist, falls als nächstes eine neue Vokabel angezeigt werden soll und `false` ist, falls als nächstes die zur aktuell angezeigten Vokabel gehörende Lösung angezeigt werden soll.

5.13.1 Benötigte Methoden

Wir erstellen eine Methode `buttonClicked(View view)`, die aufgerufen werden soll, falls auf die *Anzeigen*-Schaltfläche geklickt wird und die Methoden `neueVokabelAnzeigen()` und `loesungAnzeigen()`, die wir entsprechend der Belegung der boolschen Variable `showVoc` aufrufen:

```
1  /**
2   * Wird bei Klicken auf die Anzeigen-Schaltflaeche aufgerufen
3   * @param view
4   */
5  public void buttonClicked(View view) {
6      if(showVoc) {
7          neueVokabelAnzeigen();
8      } else {
9          loesungAnzeigen();
10     }
11 }
12
13 private void neueVokabelAnzeigen() {
14      // Neue Vokabel anzeigen
15 }
16
17 private void loesungAnzeigen() {
18     // Loesung zur aktuell angezeigten Vokabel anzeigen
19 }
```

Legen Sie zunächst fest, dass die Methode `buttonClicked` aufgerufen wird, wenn auf die *Anzeigen*-Schaltfläche geklickt wird.

5.13.2 Beschriftung der Anzeigen-Schaltfläche

Bei Aufrufen der Klasse `VocTestActivity` soll die *Anzeigen*-Schaltfläche die Beschriftung "*Abfrage starten*" haben und keine Vokabel auf der Karteikarte angezeigt werden.
Wird auf der Karteikarte aktuell die Lösung einer Vokabel angezeigt, soll die *Anzeigen*-Schaltfläche die Beschriftung "*Neue Vokabel anzeigen*" haben.

[9] Unter der Lösung einer deutschen Vokabel verstehen wir hier ihre englische Bedeutung und entsprechend bei einer englischen Vokabel ihre deutsche Bedeutung

Aufgabe 7

(a) Sorgen Sie dafür, dass bei jedem Aufruf der Activity-Klasse die Beschriftung des *Anzeigen*-Buttons auf *Abfrage starten* gesetzt wird und auf der Karteikarte nichts angezeigt wird. Überlegen Sie sich dazu, an welcher Stelle in der Activity-Klasse dies geschehen soll und auf welchen Wert Sie die Variable `showVoc` setzen müssen.

(b) Sorgen Sie dafür, dass die Beschriftung der *Anzeigen*-Schaltfläche auf *Neue Vokabel anzeigen* gesetzt wird, falls aktuell die Lösung einer Vokabel angezeigt wird. Überlegen Sie sich dazu, an welcher Stelle in der Activity-Klasse dies geschehen soll.

Um die Beschriftung "*Lösung anzeigen*" kümmern wir uns später.

5.13.3 Datenbankzugriff

Um einen zufälligen Datensatz (deutsche und englische Bedeutung einer Vokabel) aus der Datenbank auslesen zu können, benötigen wir in der Klasse `VocTestActivity` ein Objekt der Klasse `DataHandler`, das wir in der `onCreate`-Methode initialisieren:

Code 5.13: Erstellen und Initialisieren eines DataHandler-Objekts

```
1 DataHandler dbHandler;
2
3 @Override
4 protected void onCreate(Bundle savedInstanceState) {
5     super.onCreate(savedInstanceState);
6     setContentView(R.layout.activity_voctest);
7     dbHandler = new DataHandler(this);
8     ...
9 }
```

In der Klasse `DataHandler` benötigen wir eine Methode `getRandomData()`, die uns einen beliebigen Datensatz aus der Datenbank ausliest und das Wertepaar (deutsche Bedeutung, englische Bedeutung) zurückgibt.

Mit dem SQLite-Befehl

```
SELECT *
FROM vocTable;
```

erhalten wir eine Tabelle mit allen Wertepaaren; also etwa:

eins	one
zwei	two
drei	three
Universität	university
...	...

Mit dem Zusatz `ORDER BY RANDOM()` werden die Zeilen in zufälliger Reihenfolge angeordnet und mit dem Zusatz `LIMIT 1` wird nur die erste Zeile ausgewählt:

```
SELECT *
FROM vocTable
ORDER BY RANDOM()
LIMIT 1;
```

 **Aufgabe 8** 

(a) Überlegen Sie, welchen Rückgabetyp die Methode `getRandomData()` haben soll.

(b) Überlegen Sie, wie viele Elemente und wie viele Einträge in einem Element das Cursor-Objekt bei obigen SQL-Statement hat.

(c) Erstellen Sie die Methode `getRandomData()`, die ein zufälliges Vokabelpaar aus der Tabelle `vocTable` holt und zurückgibt. Nutzen Sie hier wieder die Methode `db.rawQuery(sqlstatement)`, die den durch die SQL-Anfrage spezifizierten Datensatz in einem Cursor-Objekt speichert.

5.13.4 Neue Vokabel anzeigen und Abfragerichtung

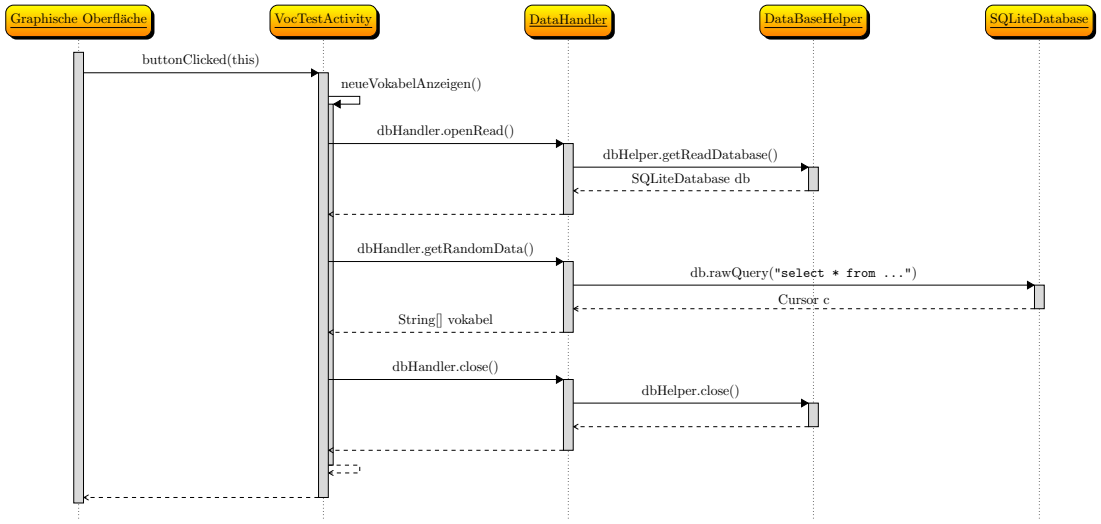
Beim Aufrufen der Methode `neueVokabelAnzeigen()` überprüfen wir zunächst, ob mindestens eine Abfragerichtung ausgewählt wurde. Mit der Methode `isChecked()` der Klasse Switch, können wir überprüfen, ob der Switch auf `ON` oder `OFF` steht.

Ist mindestens eine Abfragerichtung ausgewählt holen wir zunächst ein neues Wertepaar (deutsche Bedeutung, englische Bedeutung) aus der Datenbank.
Dazu benötigen wir Lesezugriff auf die Datenbank. Mit der Methode `getRandomData()` der Klasse DataHandler erhalten wir ein Wertepaar einer zufällig ausgesuchten Vokabel.

Ist keine Abfragerichtung ausgewählt, soll eine entsprechende Benachrichtigung angezeigt werden.

Aufgabe 9

Setzen Sie oben beschriebenes Szenario um und passen Sie die Methode `neueVokabelAnzeigen()` entsprechend an. Folgendes Sequenzdiagramm kann Ihnen dabei hilfreich sein:



Ist mindestens eine Abfragerichtung ausgewählt, müssen wir drei Fälle unterscheiden:

- beide Abfragerichtung sind ausgewählt
- es ist nur die Abfragerichtung `Deutsch` → `Englisch` ausgewählt
- es ist nur die Abfragerichtung `Englisch` → `Deutsch` ausgewählt

Aufgabe 10

Erweitern Sie die Methode `neueVokabelAnzeigen()` um eine Kontrollstruktur, mit der Sie obige drei Fälle unterscheiden können. Fügen Sie diese Kontrollstruktur bei `TODO` ein:

```
1  private void neueVokabelAnzeigen() {
2      ...
3
4      if (deutschenglisch.isChecked() || englischdeutsch.isChecked()) {
5          ...
6
7          // TODO: Kontrollstruktur einfuegen
8
9      } else {
10         ...
11     }
12 }
```

Je nach Abfragerichtung zeigen wir eine Bedeutung auf der Karteikarte an, die andere müssen wir zwischenspeichern. Dazu erstellen wir uns in der Klasse `VocTestActivity` noch eine String-Variable `loesung`. Wir kümmern uns zunächst nur um die Fälle, in denen nur eine Abfragerichtung ausgewählt wurde.
Wurde ein Eintrag in der Datenbank gefunden und ist mindestens eine Abfragerichtung ausgewählt, müssen wir unsere boolsche Variable `showVoc` auf `false` setzen, damit beim nächsten Klicken auf die *Anzeigen*-Schaltfläche die Methode `loesungAnzeigen()` aufgerufen wird. Zusätzlich ändern wir den Text der *Anzeigen*-Schaltfläche noch auf `"Lösung anzeigen"`.

 **Aufgabe 11** 

Setzen Sie oben beschriebenes Szenario um und passen Sie die Methode `neueVokabelAnzeigen()` entsprechend an.

Sind beide Abfragerichtungen ausgewählt, wollen wir zufällig entweder die deutsche oder englische Bedeutung auf der Karteikarte anzeigen und die andere Bedeutung in der Variable `loesung` speichern. Wir greifen hier auf ein Objekt `r` der Klasse `Random` zurück. Der Befehl `r.nextBoolean()` liefert als Rückgabewert zufällig entweder `true` oder `false` - man kann diesen Befehl mit einer Art Münzwurf vergleichen.
Wird `true` zurückgegeben, zeigen wir die deutsche Bedeutung auf der Karteikarte an und speichern die englische in `loesung`. Bei Rückgabewert `false` entsprechend umgekehrt.

 **Aufgabe 12** 

Wir haben Ihnen schon den Teil mit Zufallszahl erzeugen und Kontrollstruktur vorgegeben. Ergänzen Sie bei den `TODO`s noch, dass die eine Bedeutung auf der Karteikarte angezeigt wird und die andere in `loesung` gespeichert wird und fügen Sie den Programmteil dann an geeigneter Stelle in die Methode `neueVokabelAnzeigen` ein. Sie können dabei natürlich auf das String-Array `vokabel`, den String `loesung` und die View-Objekte, die Sie in Aufgabe 11 innerhalb der Methode erzeugt haben, zugreifen:

```
1  Random r = new Random();
2
3  // Erzeugen eines zufaelligen boolschen Wertes
4  if (r.nextBoolean()) {
5      // TODO
6  } else {
7      // TODO
8  }
```

5.13.5 Lösung anzeigen

Bei Aufruf `loesungAnzeigen()` müssen wir nun den Text der Karteikarte auf die in der Variable `loesung` gespeicherte Bedeutung der Vokabel setzen. Zudem setzen wir `newVoc` auf `true`:

```
1  private void loesungAnzeigen() {
2    // Loesung zur aktuell angezeigten Vokabel anzeigen
3    ((Button) findViewById(R.id.vocKarte)).setText(loesung);
4    newVoc = true;
5  }
```

5.13.6 Schriftart und -farbe

Um Lösung besser von der anderen Bedeutung der Vokabel unterscheiden zu können, setzen wir bei der Lösung den Font auf Kursiv und die Textfarbe auf blau:

```
1  /**
2  * Wird bei Klicken auf die Anzeigen-Schaltflaeche aufgerufen
3  *
4  * @param view
5  */
6  public void buttonClicked(View view) {
7
8      Button karteikarte = (Button) findViewById(R.id.vocKarte);
9
10     if (newVoc) {
11         karteikarte.setTypeface(null, Typeface.NORMAL);
12         karteikarte.setTextColor(Color.BLACK);
13         neueVokabelAnzeigen();
14         // Entsprechende Beschriftung der Anzeigen-Schaltflaeche, falls eine neue Vokabel angezeigt wird
15         ((Button) findViewById(R.id.vocButton)).setText("Loesung anzeigen");
16     } else {
17         karteikarte.setTypeface(null, Typeface.ITALIC);
18         karteikarte.setTextColor(Color.BLUE);
19         loesungAnzeigen();
20         // Entsprechende Beschriftung der Anzeigen-Schaltflaeche, falls eine Loesung angezeigt wird
21         ((Button) findViewById(R.id.vocButton)).setText("Neue Vokabel anzeigen");
22     }
23 }
```

5.13.7 Testen der Applikation

Testen Sie nun Ihre Applikation! Wir können nun unsere Vokabeln abspeichern und diese in beliebiger Richtung abfragen!

5.14 Vokabeln verwalten

Wir können nun bereits neue Vokabeln hinzufügen und die hinzugefügten Vokabeln in beliebiger Richtung abfragen. Als letzte Funktion unserer Vokabel-App möchten wir nun auch noch abgespeicherte Vokabeln editieren können.

Aufgabe 13

Ergänzen Sie dazu zunächst eine Schaltfläche *Vokabeln verwalten* in der Activity-Klasse `SelectionActivity` und geben Sie ihr die `ID voceditbutton` (s. hierzu auch das Zustandsdiagramm zur Benutzerführung auf Seite 33)

5.14.1 Die Klasse EditorActivity und ihr Layout

Aufgabe 14

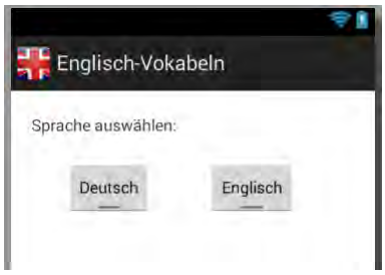
Erstellen Sie eine neue Activity-Klasse `EditorActivity` und die zugehörige Layout-Datei `activity_editor.xml` (diese wird automatisch generiert). Legen Sie fest, dass diese Activity-Klasse bei Klicken auf die Schaltfläche *Vokabeln verwalten* der Klasse `SelectionActivity` aufgerufen wird.
In der Layout-Datei benötigen wir ein `TextView`-Objekt mit dem Text *Sprache auswählen* und zwei Objekte der Klasse `ToggleButton`. Diesen weisen wir die Beschriftung *Deutsch* bzw. *Englisch* und die `ID toggledeutsch` bzw. `toggleenglisch` zu. Hier ist zu beachten, dass dies nicht wie gewohnt über die Zeile

```
android:text="@string/toggle2text"
```

geht, sondern man bei einem ToggleButton einen Text für `ON` und einen Text für `OFF` vergeben kann:

```
android:textOff="@string/toggle2text"
android:textOn="@string/toggle2text"
```

Bisher sollte Ihre Layout-Datei etwa wie folgt aussehen:



Aufgabe 15

Erstellen Sie weiter ein Objekt der Klasse `Spinner`[10] mit der ID `vocSpinner` und zwei Button-Objekte mit der Beschriftung *Vokabel ändern* bzw. *Vokabel löschen* und der `ID` `changebutton` bzw. `deletebutton`.

Die Layout-Klasse sollte nun etwa wie folgt aussehen:

5.14.2 Sprache auswählen

Es soll immer genau einer der beiden ToggleButtons ausgewählt sein, d.h. es darf nicht sein, dass sich beide im Zustand `OFF` befinden bzw. beide im Zustand `ON` befinden. Wir setzen dazu standardmäßig bei Aufruf der Activity-Klasse den ToggleButton `toggledeutsch` auf `ON` und den ToggleButton `toggleenglisch` auf `OFF`. Wir erstellen zudem eine globale boolsche Variable `deutschSelected`, die `true` sein soll, falls der ToggleButton

[10] Wir gehen später noch näher auf die Funktionalität dieser Klasse ein. Es empfiehlt sich, die Breite des Spinner-Objekts in der `XML`-Datei auf `match_parent` zu stellen.

Deutsch auf `ON` steht und `false`, falls der ToggleButton *Deutsch* auf `OFF` steht (und somit der ToggleButton *Englisch* auf `ON`)[11]:

Code 5.14: Initialisierung der ToggleButtons bei Starten der Activity

```
private boolean deutschSelected;

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

    ((ToggleButton) findViewById(R.id.toggledeutsch)).setChecked(true);
    ((ToggleButton) findViewById(R.id.toggleenglisch)).setChecked(false);
    deutschSelected = true;
}
```

 **Aufgabe 16** 

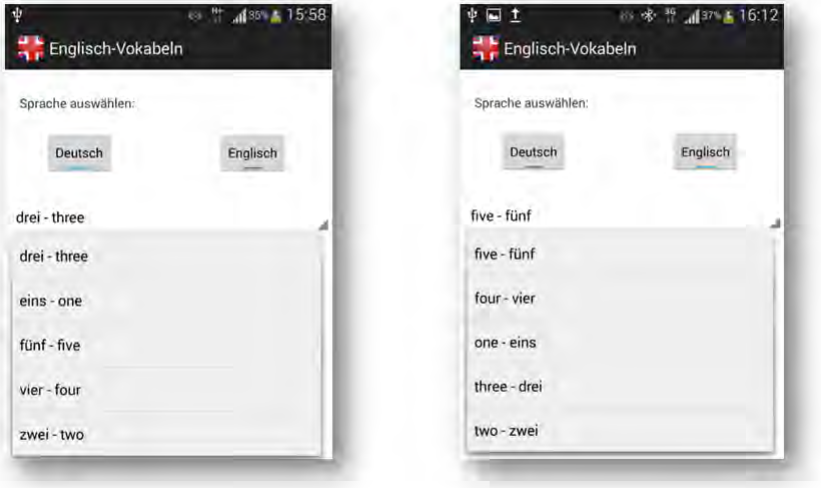
Sorgen Sie dafür, dass nicht beide `ToggleButton` gleichzeitig, oder keiner der beiden `ToggleButton` ausgewählt werden kann. Verwenden Sie hierzu unten angegebene Methode `toggleButtonClicked`, die aufgerufen werden soll, falls auf einen der beiden `ToggleButton` geklickt wird. Bei Klicken auf einen der beiden `ToggleButton` soll **immer** der andere ausgewählt werden. Wir haben Ihnen bereits eine geeignete Kontrollstruktur in der Methode vorgegeben. Vergessen Sie nicht, die Variable `deutschSelected` an geeigneten Stellen zu ändern:

```
/**
* Wird aufgerufen, wenn auf einen der beiden ToggleButton geklickt wird
*
* @param view
*/
public void toggleButtonClicked(View view) {
    // Ist bei Klicken auf einen der beiden ToggleButton bisher der deutsche ToggleButton ausgewaehlt
    if (deutschSelected) {

    // Ist bei Klicken auf einen der beiden ToggleButton bisher der englische ToggleButton ausgewaehlt
    } else {

    }
}
```

5.14.3 Methoden für Datenbankzugriff

Wir möchten in dem `Spinner`-Objekt alle abgespeicherten Vokabeln anzeigen und zwar entweder nach der deutschen Bedeutung sortiert, falls der `ToggleButton` Deutsch ausgewählt ist, oder entsprechend nach der englischen Bedeutung sortiert:

[11] Wir hätten an dieser Stelle auch eine `RadioGroup` mit zwei `RadioButton` verwenden können, die automatisch sicherstellt, dass nicht beide gleichzeitig ausgewählt sind. Dieser `RadioGroup` hätten wir in der Activity-Klasse dann einen `OnClickListener` hinzufügen müssen. Darauf haben wir an dieser Stelle bewusst verzichtet.



Folglich brauchen wir auch in dieser Activity-Klasse Zugriff auf die Datenbank, um die Vokabeln anzeigen zu können. Aus diesem Grund erzeugen wir, wie zuvor bei den anderen Activity-Klassen ein Objekt der Klasse `DataHandler` und initialisieren dieses in der `onCreate`-Methode:

```
1  private DataHandler dbHandler;
2
3  @Override
4  protected void onCreate(Bundle savedInstanceState) {
5      super.onCreate(savedInstanceState);
6      setContentView(R.layout.activity_editor);
7      dbHandler = new DataHandler(this);
8
9      ...
10 }
```

In der `DataHandler`-Klasse benötigen wir nun zwei Methoden:

- eine Methode `nachDeutscherBedeutungSortiert()`, die eine Liste aller Vokabelpaare zurückgibt, die nach den deutschen Bedeutungen sortiert ist
- eine Methode `nachEnglischerBedeutungSortiert()`, die eine Liste aller Vokabelpaare zurückgibt, die nach den englischen Bedeutungen sortiert ist

---

Aufgabe 17

Erstellen Sie eine SQL-Anfrage, die alle Vokabelpaare nach der deutschen (englischen) Bedeutung sortiert ausgibt.

Aufgabe 18

(a) Was macht unten abgebildete Methode? Was wäre der Rückgabewert, falls die Tabelle `vocTable` drei Einträge `(eins,one), (zwei,two), (drei,three)` hätte?

```
/**
* Liste aller Vokabeln, nach deutscher Bedeutung sortiert
*
* @return
*/
public ArrayList<String> nachDeutscherBedeutungSortiert() {
    ArrayList<String> list = new ArrayList<String>();
    Cursor c = db.rawQuery("select deutsch,englisch from vocTable order by deutsch;",null);

    if (c.moveToFirst()) {
        list.add(c.getString(0) + " - " + c.getString(1));

        while (c.moveToNext()) {
            list.add(c.getString(0) + " - " + c.getString(1));
        }
    }
    return list;
}
```

(b) Fügen Sie oben abgebildete Methode in der Klasse `DataHandler` ein und erstellen Sie weiter die Methode `nachenglischerBedeutungSortiert()`, die ebenfalls eine `ArrayList<String>` aller Vokabeln zurückgibt, nur dieses mal nach der englischen Bedeutung sortiert.

5.14.4 Spinner aktualisieren - die Methode `updateSpinner()`

Bei Aufruf der Activity oder welchseln zwischen den `ToggleButton`, müssen wir die Vokabelliste im `Spinner` aktualisieren. Hierzu erstellen wir uns eine Methode `updateSpinner()`, die zunächst überprüft, ob wir die Vokabeln nach deutscher oder englischer Bedeutung sortieret haben wollen und die Vokabeln anschließend je nach Sortierung als `ArrayList<String>` aus der Datenbank holt:

```
/**
* Aktualisiert die Eintraege im Spinner
*/
private void updateSpinner() {
    ArrayList<String> spinnerList;
    // Deutscher ToggleButton ausgewaehlt
    if (deutschSelected) {
        dbHandler.openRead();
        spinnerList = dbHandler.allVocOrderedByDeutsch();
        dbHandler.close();
    // englischer ToggleButton ausgewaehlt
    } else {
        dbHandler.openRead();
        spinnerList = dbHandler.allVocOrderedByEnglisch();
        dbHandler.close();
```

```
16      }
17
18      // TODO : dier haltene Liset dem Spinner zuweisen
19  }
```

Nun müssen wir bei `TODO` dem `Spinner` noch die Elemente unserer `ArrayList<String>` zuweisen. Dies geht über ein Objekt der Klasse `ArrayAdapter`:

```
1   /**
2   * Aktualisiert die Eintraege im Spinner
3   */
4   private void updateSpinner() {
5       ArrayList<String> spinnerList;
6
7       ...
8
9       Spinner spinner = (Spinner) findViewById(R.id.vocSpinner);
10
11      ArrayAdapter<String> spinnerAdapter = new ArrayAdapter<String>(this,android.R.layout.simple_spinner_item
            , spinnerList);
12      spinnerAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
13      spinner.setAdapter(spinnerAdapter);
14  }
```

 **Aufgabe 19** 

Überlegen Sie sich, an welchen Stellen in der `EditorActivity`-Klasse Sie die Methode `updateSpinner()` aufrufen müssen, um die Elemente des `Spinner`s immer aktuell und konform zu der ausgewählten Sortierung zu halten.

Testen Sie Ihre App. Sie sollten sich nun in der `EditorActivity`-Klasse alle gespeicherten Vokabeln entweder nach der deutschen oder nach der englischen Bedeutung sortiert ausgeben lassen können.

5.14.5 Vokabeln löschen

Im Spinner werden uns nun alle Vokabeln angezeigt. Wir möchten nun die im Spinner ausgewählte Vokabel löschen können. Dabei tritt jedoch folgendes Problem auf:

> *Wir müssen z.B. aus dem String "eins - one" wieder die deutsche und englische der Bedeutung der Vokabel extrahieren, da wir die Vokabel sonst nicht aus der Datenbank löschen können.*

Wir erkennen natürlich beim Hinschauen, was die deutsche und englische Bedeutung der Vokabel in diesem String ist. Doch der Computer kann dies nicht.

Was können wir also tun? Eine erste Idee wäre, den String am Bindestrich aufzuspalten und so wieder einen String mit der deutschen und einen String mit der englischen Bedeutung zu erhalten. Hier können allerdings schnell Probleme auftreten, falls in der deutschen oder englischen Bedeutung einer Vokabel auch ein Bindestrich verwendet wurde.

Eine zweite Idee wäre, sich das Vokabelpaar über die Position erneut aus der Datenbank zu holen. Der `Spinner`

bietet die Methode `getSelectedItemPosition()` an, mit der wir die Position des ausgewählten Strings im `Spinner` ermitteln können. Holen wir die Vokabeln erneut (nach deutscher oder englischer Bedeutung sortiert) aus der Datenbank, finden sich die gesuchten Bedeutungen an der gleichen Position im Cursor, wie der ausgewählte String im `Spinner`.

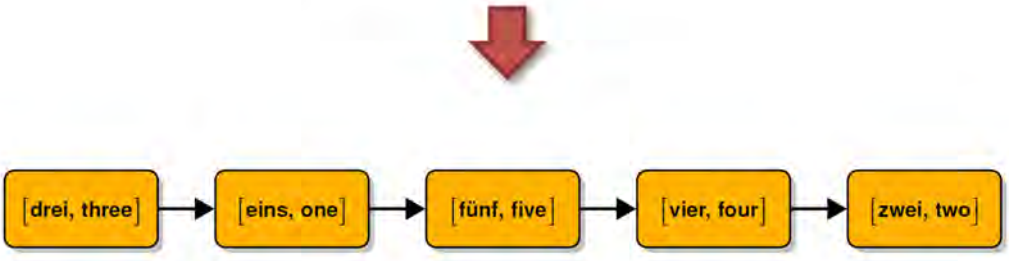
Betrachten wir hierzu folgendes Beispiel:

Angenommen, wir haben folgende Tabelle `vocTable` (links). Dann haben die Elemente im `Spinner` bei Sortierung nach deutscher Bedeutung folgende Reihenfolge (rechts):

eins	one
vier	four
fünf	five
zwei	two
drei	three

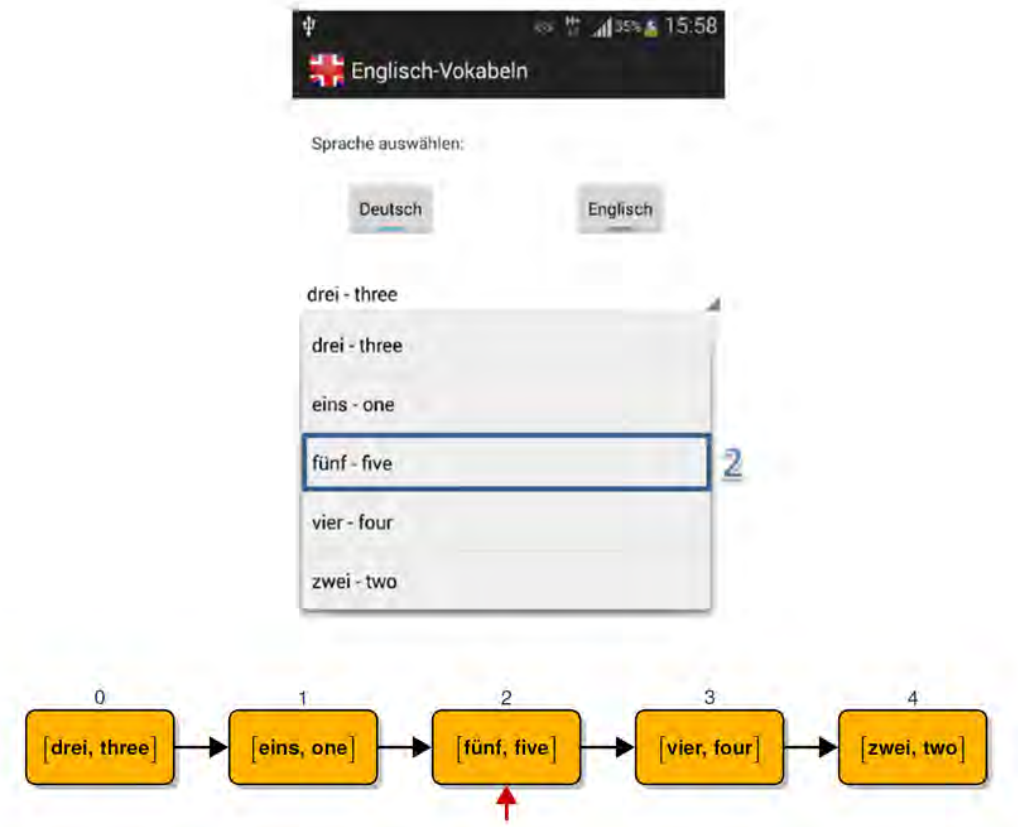
Stellen wir nun eine SQL-Anfrage an die Datenbank, alle Einträge der Tabelle `vocTable` nach deutscher Bedeutung sortiert auszugeben, erhalten wir folgendes Cursor-Objekt:

drei	three
eins	one
fünf	five
vier	four
zwei	two



Wählen wir etwa den Eintrag `"fünf - five"` im `Spinner` aus, hat dieser dort die gleiche Position (hier: 2[12]) wie das zugehörige Element im Cursor:

[12] Beachten Sie, dass wieder bei 0 angefangen wird zu zählen



Wird also auf die Schaltfläche *Vokabel löschen* geklickt, müssen wir wie folgt vorgehen:

- die Position `pos` der ausgewählten Vokabel im Spinner ermitteln
- ist Deutsch ausgewählt?
  - Vokabeln der Tabelle `vocTable` nach deutscher Bedeutung sortiert in ein Cursor-Objekt auslesen
  - Zugriff auf das (`pos`)-te Element im Cursor
- ist Englisch ausgewählt?
  - Vokabeln der Tabelle `vocTable` nach deutscher Bedeutung sortiert in ein Cursor-Objekt auslesen
  - Zugriff auf das (`pos`)-te Element im Cursor

Aufgabe 20

Erstellen Sie die Methoden `getVocByPositionDeutsch(int pos)` und `getVocByPositionEnglisch(int pos)` in der `DataHandler`-Klasse, die ein String-Array mit der deutschen und englischen Bedeutung der Vokabel zurückgeben, die an der Position `pos` im Cursor-Objekt steht, in welches alle Einträge aus der Tabelle `vocTable` nach deutscher bzw. englischer Bedeutung sortiert ausgelesen wurden.

Hinweis: Das Cursor-Objekt bietet die Methode `moveToPosition(int pos)` an, die den Zeiger auf das Element an Position `pos` im Cursor setzt.

Um eine Vokabel löschen zu können benötigen wir noch eine Methode `deleteVoc(String deuscheBed, String englischeBed)` in der Klasse `DataHandler`:

 **Aufgabe 21** 

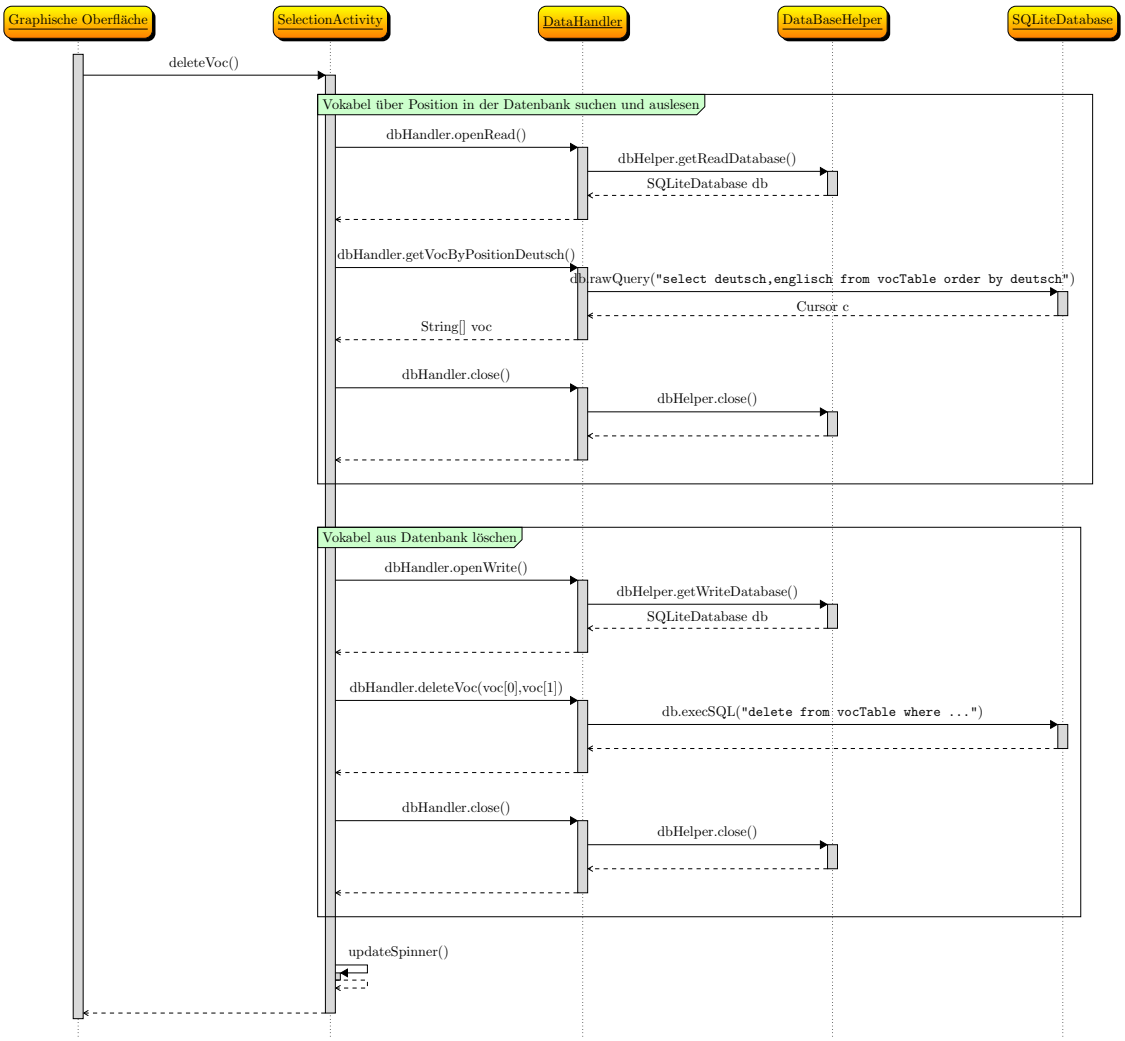
(a) Erstellen Sie ein SQL-Statement, mit dem Sie eine Vokabel über ihre deutsche und englische Bedeutung aus der Tabelle `vocTable` löschen können.

(b) Erstellen Sie die Methode `deleteVoc(String deuscheBed, String englischeBed)` in der Klasse `DataHandler`, die eine Vokabel über ihre deutsche und englische Bedeutung aus der Datenbank löscht.

Wir benötigen nun noch eine Methode `deleteButtonClicked(View view)` in der Klasse `EditorActivity`, die aufgerufen wird, wenn auf die Schaltfläche *Vokabel löschen* geklickt wird.
In dieser Methode müssen wir unterscheiden, ob die Einträge im `Spinner` nach deutscher oder englischer Bedeutung sortiert sind und holen dann die deutsche und englische Bedeutung der ausgewählten Vokabel über die Methode `getVocByPositionDeutsch(int pos)` bzw. `getVocByPositionEnglisch(int pos)` aus der Datenbank. Anschließend löschen wir die Vokabel über ihre deutsche und englische Bedeutung mit der Methode `deleteVoc(String deutscheBed, String englischeBed)` aus der Datenbank und aktualisieren den `Spinner`:

```
/**
 * Wird aufgerufen, wenn auf die Loeschen-Schaltflaeche geklickt wird
 * Loescht die im Spinner ausgewaehlte Vokabel aus der Datenbank
 * @param view
 */
public void deleteButtonClicked(View view) {
    Spinner spinner = (Spinner) findViewById(R.id.vocSpinner);
    int pos = spinner.getSelectedItemPosition();
    String[] voc;
    // ist nach deusch sortiert?
    if (deutschSelected) {
        dbHandler.openRead();
        voc = dbHandler.getVocByPositionDeutsch(pos);
        dbHandler.close();
    // oder nach englisch sortiert?
    } else {
        dbHandler.openRead();
        voc = dbHandler.getVocByPositionEnglisch(pos);
        dbHandler.close();
    }
    // Loeschen der Vokabel
    if (voc != null) {
        dbHandler.openWrite();
        dbHandler.deleteVoc(voc[0], voc[1]);
        dbHandler.close();
        // Benachrichtigung zu erfolgreichem Loeschen anzeigen
        Toast.makeText(this, "Vokabel geloescht", Toast.LENGTH_LONG).show();
   }
   // Spinner wird aktualisiert
   updateSpinner();
}
```

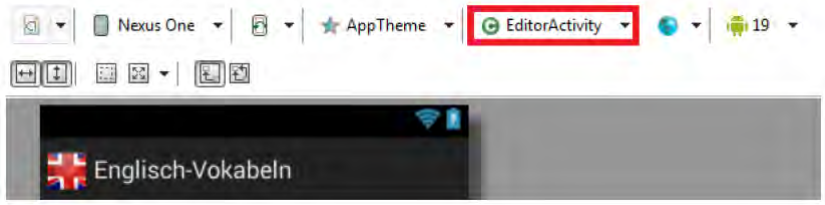
Die Methode `deleteButtonClicked` haben wir Ihnen nachfolgend auch noch in Form eines Sequenzdiagrammes dargestellt:



5.14.6 Vokabeln editieren

Bei Klicken auf die Schaltfläche *Vokabel ändern* (s. hierzu Abbildung auf Seite 53), soll ein Formular mit den bisherigen Bedeutungen der Vokabel angezeigt werden. Dabei sollen die bisherigen Bedeutungen editiert und die Änderungen an der Vokabel über eine Schaltfläche gespeichert werden können:

Für das Formular erstellen wir eine neue XML-Datei `editor_formular.xml`. Machen Sie dazu einen Rechtsklick auf den `layout`-Ordner und wählen Sie `New` → `Android XML File`. Diese XML-Datei müssen wir noch der Klasse `EditorActivity` zuweisen[13]:



Aufgabe 22

Erstellen Sie in der Layout-Datei `editor_formular.xml` zwei `TextView`- und zwei `EditText`-Objekte, sowie ein `Button-Objekt`. Geben Sie den beiden `EditText`-Objekten die `IDs` `editdeutscheBed` bzw. `editenglischeBed` und dem `Button`-Objekt die `ID` `saveButton`.
Sie können sich hierzu an obiger Abbildung orientieren.

ContentView wechseln

Bei Klicken auf die Schaltfläche *Vokabel ändern* (wir erstellen hierzu die Methode `editButtonClicked(View view)` in der `EditorActivity`-Klasse) möchten wir nun anstelle der bisherigen Layout-Datei `activity_editor.xml` die neu angelegte Layout-Datei `editor_formular.xml` anzeigen. Mit dem Befehl `setContentView(R.layout.editor_formular)` können in der `EditorActivity`-Klasse die angegebene Datei als neues Layout festlegen.

Bevor wir die Layout-Datei wechseln, müssen wir noch die deutsche und englische Bedeutung der aktuell ausgewählten Vokabel zwischenspeichern, um diese in der neuen Layout-Datei anzeigen zu können bzw. zu wissen, welche Vokabel editiert werden soll.
Die Bedeutungen der ausgewählten Vokabel können wir wie vorher beim Löschen über die beiden Methoden `getVocByPositionDeutsch(int pos)` bzw. `getVocByPositionEnglisch(int pos)` aus der Datenbank auslesen.

```
1  private String altDeBed;
2  private String altEnBed;
3
4  /**
5   * Wird aufgerufen, wenn auf die Editieren-Schaltflaeche geklickt wird
6   * Aendert die Layout-Datei und zeigt die ausgewaehlte Vokabel zum Editieren
7   * an
8   *
9   * @param view
10  */
11 public void editVoc(View view) {
```

[13] s. hierzu auch das Zustandsdiagramm zur Benutzerführung auf Seite 33

```
12      Spinner spinner = (Spinner) findViewById(R.id.vocSpinner);
13      String[] voc;
14      int pos = spinner.getSelectedItemPosition();
15
16      // ist nach deutsch sortiert?
17      if (deutschSelected) {
18          dbHandler.openRead();
19          voc = dbHandler.getVocByPositionDeutsch(pos);
20          dbHandler.close();
21          // oder nach englisch sortiert?
22      } else {
23          dbHandler.openRead();
24          voc = dbHandler.getVocByPositionEnglisch(pos);
25          dbHandler.close();
26      }
27      // Aendern der Layout-Datei
28      setContentView(R.layout.editor_formular);
29      // Alte Bedeutungen der Vokabel zwischenspeichern (wichtig fuer SQL-Anfrage)
30      altDeBed = voc[0];
31      altEnBed = voc[1];
32
33      // alte Bedeutungen werden zum Editieren in den Eingabefeldern angezeigt
34      ((EditText) findViewById(R.id.editdeutscheBed)).setText(voc[0]);
35      ((EditText) findViewById(R.id.editenglischeBed)).setText(voc[1]);
36  }
```

Die Methode `saveChangeButtonClicked`

Wird nun auf die *Ändern*-Schaltfläche im Formular geklickt und sind beide Textfelder nicht leer, soll die bisherige Vokabel mit den neuen Bedeutungen überschrieben werden.

 **Aufgabe 23** 

(a) Erstellen Sie zunächst eine SQL-Anfrage, die eine Vokabel mit der bisherigen deutschen Bedeutung `altDeBed` und englischen Bedeutung `altDeBed` mit der neuen deutschen Bedeutung `neuDeBed` und englischen Bedeutung `neuEnBed` überschreibt.

(b) Erstellen Sie eine Methode `updateVoc(String altDeBed, String altEnBed, String neuDeBed, String neuEnBed)` in der `DataHandler`, in der obige SQL-Anfrage ausgeführt wird.

Wir benötigen noch eine Methode `saveChangeButtonClicked(View view)`, die aufgerufen wird, wenn in unserem Formular auf die *Ändern* Schaltfläche geklickt wird. In dieser Methode, lesen wir die in den `EditText`-Objekten eingegebene neue deutsche bzw. neue englische Bedeutung aus und überschreiben die alten Bedeutungen der Vokabel (diese sind in den Variablen `altDeBed` und `altEnBed` gespeichert) mit den neuen Bedeutungen, falls sowohl bei der neuen deutschen als auch neuen englischen Bedeutung Text eingegeben wurde. Nach der Änderung geben wir eine Benachrichtigung aus und wechseln die ContentView wieder zurück auf die Layout-Datei `activity_editor` und aktualisieren den `Spinner`:

```
/**
 * Wird aufgerufen, wenn auf die Aendern-Schaltflaeche im Editor-Formular geklickt wird
 * @param view
 */
public void saveChangeButtonClicked(View view) {
    EditText editDeutsch = (EditText) findViewById(R.id.editdeutscheBed);
    EditText editEnglisch = (EditText) findViewById(R.id.editenglischeBed);

    // Auslesen der neuen Bedeutungen der zu editierenden Vokabel aus den Eingabefeldern
    String neuDeBed = editDeutsch.getText().toString();
    String neuEnBed = editEnglisch.getText().toString();

    // war eines der beiden Eingabefelder leer ...
    if (neuDeBed.equals("") || neuEnBed.equals("")) {
        // ... wird eine Fehlermeldung ausgegeben
    Toast.makeText(this, "Bitte beide Felder ausfuellen",Toast.LENGTH_LONG).show();
    // falls nicht ...
    } else {
        // wird die Vokabel in der Datenbank ueberschrieben
        dbHandler.openWrite();
        dbHandler.updateVoc(altDeBed, altEnBed, neuDeBed, neuEnBed);
        dbHandler.close();

        Toast.makeText(this, "Vokabel geaendert", Toast.LENGTH_LONG).show();
        // Aendern der Layout-Datei
        setContentView(R.layout.activity_editor);
        // Spinner wird aktualisiert
        updateSpinner();
    }
}
```

Unsere Vokabel-App ist nun fertig. Lade die App auf dein Handy und teste die neu hinzugekommenen Funktionen!

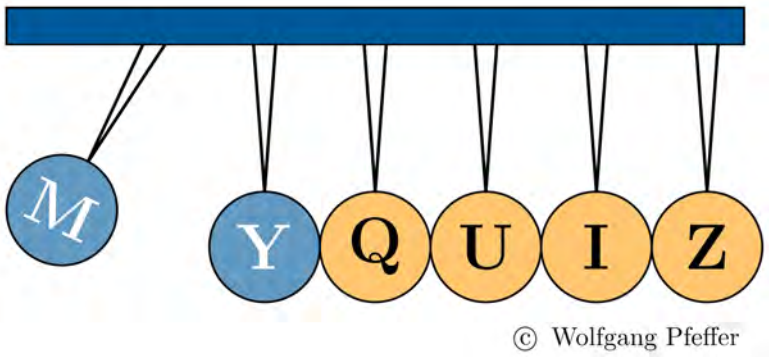
Kapitel 6

MyQuiz-App

Die Englisch-Vokabel-App aus Kapitel 5 war bewusst stark geleitet, da an ihr der Aufbau einer etwas umfangreicheren Aufgabenstellung, sowie das Wechseln zwischen Activity-Klassen, der Wechsel von Layout-Dateien, Verwendung und Zugriff auf einer app-interne Datenbank etc. aufgezeigt werden sollte.
Die Applikation, die in diesem Kapitel erstellt werden soll, ist von der Funktionalität und Anforderung sehr ähnlich zu der Englisch-Vokabel-App und wird aus diesem Grund kaum angeleitet. Sie sollen sich anhand der Aufgabenstellung überlegen, wie Sie die App umsetzen, wie viele Activity-Klassen sie benötigen, usw.
Im Lösungskapitel finden Sie einen möglichen Aufbau sowie die Komplett-Lösung, allerdings ohne Ausführungen. Falls Sie an manchen Stellen Probleme haben sollten, empfehlen wir, noch mal bei der Englisch-Vokabel-App nachzuschlagen, wie das entsprechende Szenario dort umgesetzt wurde.
Viel Erfolg!

6.1 Aufgabenstellung

Wir möchten eine Quiz-App programmieren. Dabei soll es möglich sein, Fragen mit vier Antwortmöglichkeiten abzuspeichern, wobei eine davon richtig ist. Über eine andere Schaltfläche soll das Spiel gestartet werden können, wobei zufällig eine der abgespeicherten Datensätze (Frage plus zugehörige Antworten) ausgewählt und angezeigt werden soll. Falls der Benutzer die Frage richtig beantwortet, wird ihm die nächste Frage angezeigt, usw. Anhand eines Zählers wird dem Benutzer angezeigt, wie viele Fragen er bereits richtig beantwortet hat (Sie können natürlich auch Gewinnstufen ähnlich zu Wer wird Millionär einbauen). Wird eine Frage falsch beantwortet, soll der Zähler zurückgesetzt werden und das Quiz startet von vorne. Über eine dritte Schaltfläche soll es möglich sein, die Datensätze zu verwalten, d.h. entweder Datensätze zu löschen oder zu bearbeiten.
Sie können Ihre App natürlich durch beliebige Zusatzfunktionen erweitern (z.B. Integrieren einer Highscore-Anzeige, Joker, etc.).



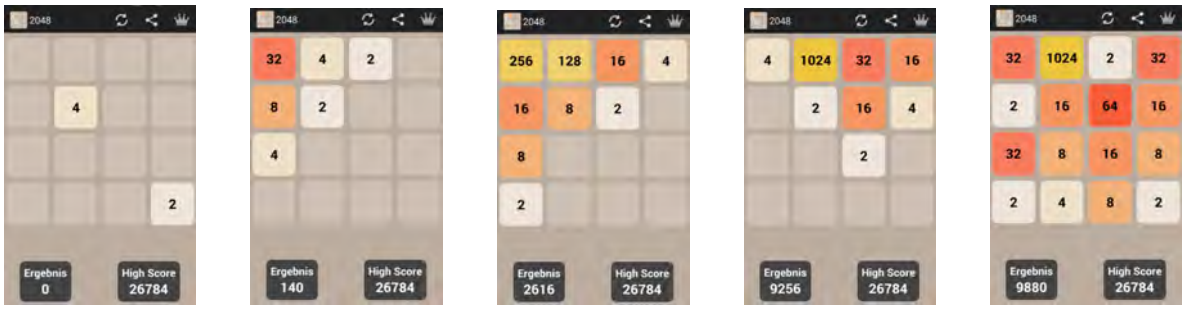
Kapitel 7

My2048-App

Die Aufgaben bisher waren sehr stark auf den Aufbau von Java-Android-Projekten fokusiert. Besonders im Fokus lag das Zusammenwirken zwischen Layout-Dateien und Activity-Klassen, die Ereignisverarbeitung sowie das Wechseln zwischen Activity-Klassen. Die Funktionalität der Apps stand bisweilen eher im Hintergrund, komplexere Kontrollstrukturen oder Algorithmen spielten keine Rolle.
In diesem Kapitel soll eine App erstellt werden, die genau das Gegenteil bietet. Der Aufbau der App ist mit einer Activity-Klasse und zugehöriger Layout-Datei nahezu minimal. Dafür benötigen wir einige verschiedene Datentypen, wie z.B. zweidimensionale Arrays und doppelt verkettete Listen. Somit erfordert diese Aufgabe grundlegende Kenntnisse hinsichtlich verschiedener Datenstrukturen und vor allem von Kontrollstrukturen.

7.1 Die App 2048

Anders als bei den bisherigen Aufgaben dient uns hier eine App als Vorlage, die es im Google Play Store gibt und mit über 10 Millionen Downloads, sowie knapp 700.000 Bewertungen zudem sehr beliebt ist.
Wir empfehlen, die App aus dem Play Store zunächst ein paar mal zu spielen, bevor man sich an die Erstellung seiner eigenen Version macht. Im Folgenden ein paar Screenshots des Spiels:



Spielanleitung:

Das Spiel beginnt mit zwei Feldern, die jeweils den Wert 2 tragen. Ziel ist es, zwei Kacheln mit derselben Nummer x über die Pfeiltasten zusammenzubringen und so zu einer Kachel zu vereinen, die den Wert $2x$ hat. Aus zwei Kacheln wir eine mit dem doppelten Wert und ihr bekommt diesen Wert als Punkte gutgeschrieben. Nach jedem Spielzug taucht ein neues Spielfeld mit dem Wert 2 oder 4 auf, dass es ebenfalls unterzubringen gilt. Je höher der Wert der vereinten Kacheln, umso schneller wächst der Punktestand. Ziel ist es, in einem Feld den

Wert 2048 zu erreichen. Hierfür muss das Feld insgesamt elf Mal verdoppelt werden ($2048 = 2^{11}$).
Es gibt vier mögliche Züge: nach links, rechts, oben oder nach unten wischen. Angenommen wir wischen nach links. Dann werden die Kacheln in *allen* vier Zeilen so weit nach links wie möglich geschoben. Treffen zwei gleiche Kacheln aufeinander, verschmelzen sie zu einer Kachel mit dem doppelten Wert. Die neu entstandene Kachel kann in diesem Zug nicht noch mal verschmelzen. Es können mehr Kacheln in einem Zug verschmelzen.

Beachten Sie, dass nach einem Zug immer eine neue Kachel erscheint! Entsprechend sind Züge nach rechts, oben und unten möglich. Das Spiel endet, wenn eine Kachel den Wert 2048 annimmt oder falls kein Zug mehr möglich ist.

7.2 Activity-Klasse und Layout

Für die App benötigen wir nur eine Activity-Klasse mit zugehöriger Layout-Datei. Wir haben diese `GameActivity` genannt. Für das Spielfeld verwenden wir ein `GridLayout` und 16 Button-Objekte, denen wir später entsprechend der Nummer auf der Kachel einen anderen Hintergrund zuweisen. Wir werden folgende Hintergründe verwenden:

Weiter benötigen wir noch zwei TextView-Objekte für das Anzeigen des aktuellen Punktestands bzw. des Highscores sowie eine Schaltfläche, mit der wir ein neues Spiel starten können. Das Layout der Activity-Klasse sollte ungefähr wie folgt aussehen:

Wir fassen unser Spielfeld als 4×4-Matrix auf und geben den Button-Objekten dementsprechend folgende IDs:

b11	b12	b13	b14
b21	b22	b23	b24
b31	b32	b33	b34
b41	b42	b43	b44

Intern fassen wir unser Spielfeld als zweidimensionales Integer-Array auf. Die Button-Objekte speichern wir uns in der Activity-Klasse auch in einem zweidimensionalen Button-Array:

Code 7.1: MainActivity.java

```
1  package com.example.administrator.my2048;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.widget.Button;
6
7  /**
8  * Created by Wolfgang Pfeffer on 08.04.2015.
9  */
10 public class GameActivity extends Activity {
11     // Klassenattribute fuer die interne Repraesentation des Spielfelds als zweidimensionales Integer-Array
           bzw. den Button-Objekten als zweidimensionales Button-Array
12     private Integer[][] matrix;
13     private Button[][] buttons;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_game);
19         // Speichern der Button-Objekte in einem zweidimensionalen Button-Array
20         Button[][] viewbuttons = {{(Button) findViewById(R.id.b11), (Button) findViewById(R.id.b12),(Button)
               findViewById(R.id.b13), (Button) findViewById(R.id.b14)},{(Button) findViewById(R.id.b21), (
               Button) findViewById(R.id.b22),(Button) findViewById(R.id.b23), (Button) findViewById(R.id.b24)
               },{(Button) findViewById(R.id.b31), (Button) findViewById(R.id.b32),(Button) findViewById(R.id.
               b33), (Button) findViewById(R.id.b34)},{(Button) findViewById(R.id.b41), (Button) findViewById(
               R.id.b42),(Button) findViewById(R.id.b43), (Button) findViewById(R.id.b44)}};
21         buttons = viewbuttons;
22         // Initialisieren des zweidimensionalen Integer-Arrays mit 4 Zeilen und 4 Spalten
23         matrix = new Integer[4][4];
24     }
25 }
```

7.3 Detektieren von Wischen

Uns ist kein vordefinierter Listener für das Detektieren von Wischen bekannt. Nach Recherche im Internet haben wir uns für folgende Lösung entschieden:

- wir erstellen eine Klasse `SimpleGestureFilter` mit vorgegebenen Code, die noch das Interface `SimpleGestureListener` beinhaltet. In dieser Klasse ist das Detektieren einer Wischbewegung, sowie das Detektieren eines Doppelklicks (was wir nicht brauchen werden) gekapselt
- unsere Activity-Klasse implementiert das Interface `SimpleGestureListener` der Klasse `SimpleGestureFilter` und hat folglich die beiden Methode `onSwipe(int direction)` und `onDoubleTap()`
- wir erzeugen in unserer Activity-Klasse ein Attribut der Klasse `SimpleGestureFilter`, das wir in der `onCreate()`-Methode initialisieren
- über den Aufrufparameter `direction` können wir zwischen den vier Wischbewegungen unterscheiden. Es gibt die Fälle `SimpleGestureFilter.SWIPE_RIGHT, SimpleGestureFilter.SWIPE_LEFT, SimpleGestureFilter.SWIPE_UP, SimpleGestureFilter.SWIPE_DOWN`

Die Klasse `SimpleGestureFilter` (verwenden wir als Blackbox) finden Sie im Kapitel Lösung in Abschnitt 11.4.1.

Wir müssen in unserer Activity-Klasse noch folgende Änderungen vornehmen:

Code 7.2: MainActivity.java

```
1  package com.example.administrator.my2048;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.widget.Button;
6  import com.example.administrator.my2048.SimpleGestureFilter.SimpleGestureListener;
7
8  /**
9  * Created by Wolfgang Pfeffer on 08.04.2015.
10 */
11 public class GameActivity extends Activity implements SimpleGestureListener {
12     ...
13     private SimpleGestureFilter detector;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_game);
19         ...
20         detector = new SimpleGestureFilter(this,this);
21     }
22
23     @Override
24     public void onSwipe(int direction) {
25     }
26
27     @Override
28     public void onDoubleTap() {
29     }
30 }
```

Denken Sie daran, mittels `implements SimpleGestureListener` festzulegen, dass die Activity-Klasse das Interface implementieren muss. Die Methoden `onSwipe` und `onDoubleTap` können Sie anschließend mit dem Tastekürzel `Strg+I` (Android Studio) automatisch einfügen lassen.

In der Methode `onSwipe` können wir gleich noch die vier Fälle unterscheiden, auf denen wir später entsprechend reagieren müssen:

Code 7.3: Die Methode `onSwipe(int direction)`

```
@Override
public void onSwipe(int direction) {
    switch (direction) {
        case SimpleGestureFilter.SWIPE_RIGHT:
            //TODO
            break;
        case SimpleGestureFilter.SWIPE_LEFT:
            //TODO
            break;
        case SimpleGestureFilter.SWIPE_UP:
            //TODO
            break;
        case SimpleGestureFilter.SWIPE_DOWN:
            //TODO
            break;
    }
}
```

7.4 Erstellen von wichtigen Hilfsmethoden

In diesem Abschnitt werden wir wichtige Hilfsmethoden erstellen, die wir später auf alle Fälle benötigen werden.

7.4.1 Matrix initialisieren

Wir benötigen eine Hilfsmethode `void initialisiereMatrix()`, die alle Einträge unseres Integer-Arrays `matrix` auf 0 (Default-Wert) setzt.

7.4.2 Beschriftung und Hintergrund der Button-Objekte aktualisieren

Wir benötigen eine Hilfsmethode `void updateView()`, die die Beschriftung der Button-Objekte auf die im Integer-Array `matrix` gespeicherten Werte setzt. Weiter muss je nach Wert auch der Hintergrund der Button-Objekte geändert werden.

7.4.3 Punktestand aktualisieren

Wir benötigen eine Hilfsmethode `void updateScore(int value)`, die den aktuellen Punktestand um einen im Aufrufparameter übergebenen Wert `value` erhöht und den neuen Wert anzeigt.

7.4.4 Prüfen ob noch ein Zug möglich ist

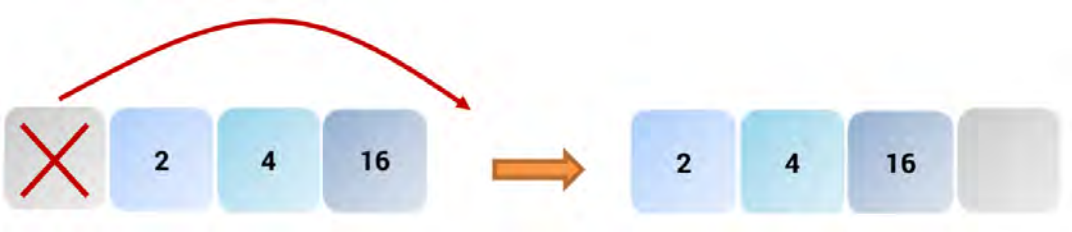
Wir benötigen eine Hilfsmethode `boolean zugMoeglich()`, mit der wir, falls es keine leeren Felder mehr gibt, prüfen können, ob noch ein oder mehrere Züge möglich sind. Überlegen Sie sich dazu, wann noch ein Zug möglich ist bzw. welche Fälle Sie dabei überprüfen müssen.

7.4.5 Neuen Wert an freier Stelle einfügen

Wir benötigen eine Hilfsmethode `void neuerWert()`, die an einer zufällig ausgewählten, freien Stelle in der Matrix `matrix` einen neuen Wert einfügt. Dieser Wert soll zu 80 Prozent die Zahl 2 sein und zu 20 Prozent die Zahl 4. Gleichzeitig soll in dieser Methode überprüft werden, ob das Spiel ggf. vorbei ist (alle Felder sind belegt und es ist kein Zug mehr möglich). In diesem Fall soll zunächst nur ein Toast ausgegeben werden und eine boolsche Variable `finished`, die mit `false` initialisiert wurde, auf `true` gesetzt werden.

7.5 Vorgehensweise beim Wischen

Als nächstes müssen wir uns überlegen, welche Schritte wir vornehmen müssen, falls der Benutzer nach links, rechts, oben bzw. unten wischt. Wir beschränken uns zunächst auf das Wischen nach links. In diesem Fall müssen wir die einzelnen Zeilen der Matrix durchgehen und alle Elemente so weit nach links wie möglich zu schieben bzw. benachbarte Elemente zusammenzufassen.
Ist zum Beispiel das erste Feld einer Zeile leer, müssen alle anderen Felder um eins nach links gerutscht werden, d.h. eigentlich nichts anderes, dass das leere Feld gelöscht wird und rechts wieder angehängt wird:



Da solche Umformungen in einem Array nicht so ohne weiteres möglich sind, haben wir uns dazu entschieden, uns die einzelnen Zeilen in eine verkettete Liste (`LinkedList`) umzuwandeln und die Umformungen an dieser vorzunehmen. Nach den Umformungen aktualisieren wir dann die Zeile in unserem zweidimensionalen Integer-Array `matrix`.

7.5.1 Hilfsmethoden für das Umwandeln einer bestimmten Zeile bzw. Spalte in eine verkettete Liste

Wir benötigen also zwei Hilfsmethoden `LinkedList<Integer> getRowAsList(int rownumber)` bzw. `LinkedList<Integer> getColAsList(int colnumber)`, die uns die Zeile `rownumber` bzw. Spalte `colnumber` (benötigen wir später für nach oben bzw. unten wischen) unseres zweidimensionalen Integer-Arrays `matrix` als `LinkedList<Integer>` zurückgibt.

7.5.2 Methode, die aufgerufen wird, wenn nach links gewischt wird

Wir stellen eine Methode `boolean swipeLeft()`, die aufgerufen wird, falls der Benutzer nach links gewischt hat. Im Rückgabewert vom Typ `boolean` teilen wir mit, ob es Änderungen in unserer Matrix gab oder nicht. Es soll nämlich nur dann soll ein neuer Wert eingefügt werden, falls sich beim Wischen auch was verändert hat, d.h. Wischen nach links überhaupt möglich war. Um festzustellen, ob sich was verändert hat, erstellen wir bei Methodenanfang eine echte Kopie von `matrix`[1] und vergleichen diese am Ende mit dem durch die Operationen entstandenen zweidimensionalen Integer-Array.
Unsere Methode holt sich nun Schritt für Schritt jede Zeile unseres zweidimensionalen Integer-Arrays als Liste und führt die entsprechenden Operationen darauf aus. Anschließend wird die entsprechende Zeile in der Matrix aktulaisiert:

Code 7.4: Die Methode `boolean swipeLeft()`

```
1 /**
2 * Wird aufgerufen, wenn nach links gewischt wird
```

[1] Wir verweisen hier auf die Problematik in Java, ein zweidimensionales Array zu kopieren, ohne nur einen Verweis auf das Originalprojekt zu bekommen.

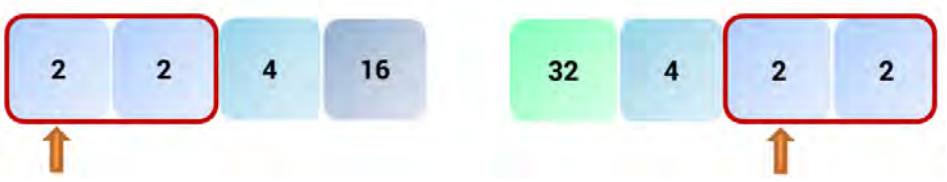
```
3  * @return
4  */
5  private boolean swipeLeft() {
6      Integer[][] copy = copyOfMatrix();
7      LinkedList<Integer> list;
8
9      for (int i = 0; i < 4; i++) {
10         list = getRowAsList(i);
11         // TODO Operationen durchfuehren und matrix aktualisieren
12     }
13
14     return Arrays.deepEquals(copy,matrix);
15 }
16
17 /**
18 * Erstellt eine echte Kopie des zweidimensionalen Integer-Arrays matrix
19 * @return
20 */
21 private Integer[][] copyOfMatrix() {
22     Integer[][] copy = new Integer[matrix.length][];
23     for (int i = 0; i < 4; i++) {
24         copy[i] = matrix[i].clone();
25     }
26     return copy;
27 }
```

Welche Fälle müssen unterschieden werden?

- Es kann ein Feld ohne Eintrag gefunden werden:

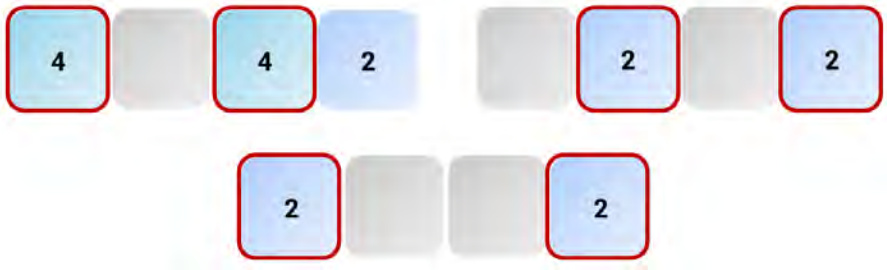
  Überlegen Sie sich, welche Operationen Sie in diesem Fall mit der Liste durchführen müssen, und welches Element Sie in der Liste als nächstes betrachten müssen.
  Achten Sie auch auf den Fall, dass eine Zeile nur aus leeren Feldern bestehen kann!

- Wird ein Feld mit Eintrag gefunden, kann es folgende Möglichkeiten geben:
  - Das benachbarte Feld hat den gleichen Eintrag:



    Überlegen Sie sich, welche Operationen Sie in diesem Fall mit der Liste durchführen müssen, und welches Element Sie als nächstes betrachten müssen, denn es kann ja auch sein, dass in einer Zeile zwei Elemente zusammengefasst werden müssen:

– Es kann auch nicht direkt benachbarte Felder geben, die zusammengefasst werden müssen:



Überlegen Sie sich, welche Operationen Sie in diesen Fällen durchführen müssen, und welche Element Sie als nächstes betrachten müssen.

Setzen Sie die oben beschriebenen Fälle in der Methode `swipeLeft()` um und denken Sie auch daran, das zweidimensionale Integer Array `matrix` an geeigneter Stelle zu aktualisieren. Sie können sich hierfür etwa eine Methode `void updateRowInMatrix(int rownumber)` erstellen.

7.5.3 Methoden, die aufgerufen werden, wenn nach rechts, oben bzw. unten gewischt wird

Erstellen Sie nun, ausgehend von der Methode für nach links wischen entsprechende Methoden für die anderen drei Fälle. Sie können sich im Falle von nach rechts wischen an obigen Szenario orientieren.
Überlegen Sie sich weiter, ob Sie für das Wischen nach oben bzw. unten eigene Methoden benötigen, oder ob Sie diese beiden Fälle nicht in die Methoden von nach links bzw. rechts wischen integrieren können.

7.6 Score aktualisieren

Immer wenn Felder zusammengefasst werden, erhöht sich der Score um die neu generierte Zahl[2].
Ist der Score größer als der aktuelle Highscore, wird auch die Highscore-Anzeige aktualisiert.

Setzen Sie oben beschriebenes Szenario in Ihrer App um!

7.7 Persistentes Speichern der Daten

Es ist natürlich wünschenswert, dass der erzielte Highscore auch nach Beenden der App bzw. Neustarten des Smartphones erhalten bleibt. Genauso soll es möglich sein, ein angefangenes Spiel später weiterspielen zu können. Wir benötigen folglich eine Datenbankanbindung, um die Einträge der Felder, sowie den aktuellen Highscore abzuspeichern. Erstellen Sie dazu eine Datenbank `gameDB` mit der Tabelle `gameTable`. Falls Sie Schwierigkeiten haben, eine Datenbank in Ihre App zu integrieren, können Sie im Kapitel 5 nachlesen. Überlegen Sie sich

- wie viele Spalten Ihre Tabelle haben soll! Eine Möglichkeit wäre zum Beispiel, für jede Zeile eine Spalte anzulegen

[2] *Beispiel:* Verschmelzen zwei Blöcke mit der Zahl 4 zu einem Block mit der Zahl 8, erhöht sich der Score um 8.

- welche Methoden Sie für den Zugriff auf die Datenbank benötigen
- an welcher Stelle Sie in Ihrem Programm die aktuellen Einträge abspeichern (es gibt z.B. die Methode `onPause()`, die automatisch aufgerufen wird, falls die App pausiert wird)
- an welcher Stelle Sie in Ihrem Programm überprüfen, ob Daten abgespeichert sind und diese in den entsprechenden Feldern anzeigen müssen

7.8 Spielen

Versuchen Sie, einen Block mit dem Wert 2048 zu erreichen!

Kapitel 8

Würfelbecher-App

Überblick:

Um einen besseren Vergleich zwischen der Entwicklung von Apps mit dem AppInventor und Java Android ziehen zu können, werden wir in diesem und nächsten Kapitel zwei Aufgaben vorstellen, die bereits in dem Modul „App-Entwicklung auf Mobile Devices“ mit dem AppInventor bearbeitet wurden.
Falls Sie bereits die Vokabel-App Aufgabe aus Kapitel 5 bearbeitet haben, sollten Sie bei dieser Aufgabe, außer mit der Interaktion mit dem Beschleunigungssensor, keine Probleme haben. Wir werden hier allerdings dennoch eine möglichst ausführliche Lösung anbieten.

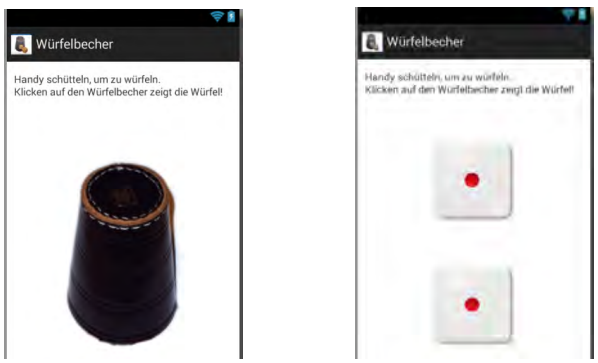


Abbildung 8.1: Die zwei verschiedenen Zustände der Würfelbecher-App. Während des Würfeln wir nur der Würfelbecher angezeigt (links). Beim „Hochheben“ des Bechers werden die Würfelergebnisse angezeigt (rechts).

8.1 Aufgabenstellung

Zwei Würfel sollen gewürfelt werden, indem man das Handy schüttelt. Während man würfelt, sieht man nur den Würfelbecher und man hört ein Würfelgeräusch. Wenn man den Würfelbecher „hochhebt“ (berührt), dann erscheinen die Würfel.

8.2 Projekt anlegen

Zunächst müssen wir wieder ein Android-Applikation-Projekt anlegen. Wir gehen hier genauso vor, wie in Kapitel 4 in Abschnitt 4.1. Als App-Namen verwenden wir etwa *Würfelbecher* und als App-Symbol ein geeignetes Bild:

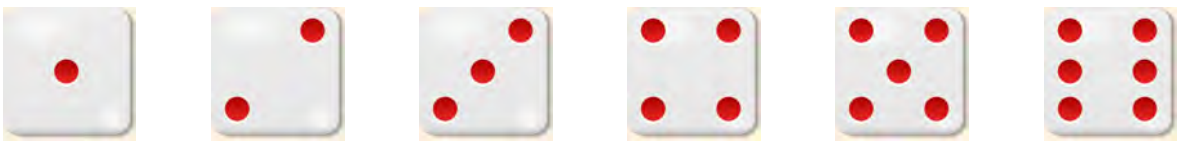
8.3 MainActivity und Layout

Zunächst ändern wir die Klasse `MainActivity.java`[1] leicht ab, da wir einige generierte Einstellungen nicht benötigen:

Code 8.1: MainActivity.java

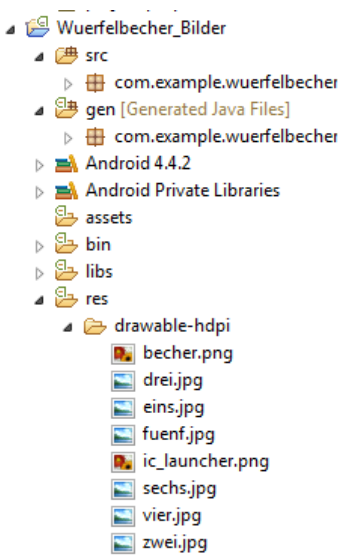
```
1  package com.example.wuerfelbecher;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class MainActivity extends Activity {
7
8    @Override
9    protected void onCreate(Bundle savedInstanceState) {
10     super.onCreate(savedInstanceState);
11     setContentView(R.layout.activity_main);
12   }
13 }
```

Im Gegensatz zum AppInventor wollen wir hier anstatt normaler Ziffern Bilder der Würfelergebnisse anzeigen lassen. Dazu benötigen wir zunächst möglichst einheitliche Bilder der sechs Seiten eines Würfels:



[1] `Würfelbecher → src → com-example.wuerfelbecher`

Diese Bilder und das Bild des Würfelbechers speichern wir wie gewohnt im Ordner `res/drawable-hdpi` im Package-Explorer:



Nun können wir das Layout festlegen, das beim Starten der Applikation angezeigt werden soll. Das Layout sollte etwa wie folgt aussehen:

Für den Würfelbecher haben wir ein Button-Objekt verwendet. Weiter haben wir zwei TextView-Objekte für die Würfel verwendet, die allerdings zunächst unsichtbar gemacht wurden. Dies geht mit dem Befehl:

```
android:visibility="invisible"
```

Unsere Layout-Datei sieht bisher wie folgt aus:

Code 8.2: activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:background="@android:color/white"
    android:layout_height="match_parent"
    tools:context="com.example.wuerfelbecher.MainActivity"
    tools:ignore="MergeRootFrame">

    <TextView
        android:id="@+id/wuerfel1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:text="@string/wuerfel1_text"
        android:layout_marginTop="114dp"
        android:textSize="75sp"
        android:visibility="invisible" />

    <TextView
        android:id="@+id/wuerfel2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/wuerfel1"
        android:layout_below="@+id/wuerfel1"
        android:layout_marginTop="62dp"
        android:text="@string/wuerfel2_text"
        android:textSize="75sp"

        android:visibility="invisible" />

    <TextView
        android:id="@+id/textanzeige"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/wuerfelbecher"
        android:layout_alignParentTop="true"
        android:layout_marginTop="16dp"
        android:text="@string/textanzeige_text"
        android:visibility="visible" />

    <Button
        android:id="@+id/wuerfelbecher"
        android:layout_width="300dp"
        android:layout_height="300dp"
        android:layout_alignTop="@+id/wuerfel1"
        android:layout_centerHorizontal="true"
        android:background="@drawable/becher"
        android:onClick="onBecherClick"
        android:visibility="visible" />

</RelativeLayout>
```

8.4 Initialisieren

Im nächsten Schritt erstellen wir entsprechende Variablen für die erstellten View-Objekte und weisen diesen die entsprechenden Objekte zu. Dies geht wie gewohnt über die Methode `findViewById`. Wichtige dabei ist, dass wir auf das entsprechende Objekt „casten“, d.h. dass wir festlegen, was für ein Objekt welcher Klasse wir mit der angegebenen `ID` finden:

Code 8.3: Erstellen und Initialisieren von Variablen für die erzeugten View-Objekte.

```
/*
 * Variablen fuer die benoetigten View-Objekte
 */
private Button wuerfelbecher;
private TextView wuerfel1;
private TextView wuerfel2;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Initalisieren der benoetigten View-Objekte
    wuerfelbecher = (Button) findViewById(R.id.wuerfelbecher);
    wuerfel1 = (TextView) findViewById(R.id.wuerfel1);
    wuerfel2 = (TextView) findViewById(R.id.wuerfel2);
```

Weiter erstellen wir noch zwei Variablen `ergebnis1` und `ergebnis2` für die Würfelergebnisse und eine Variable `becherVisible`, in der wir speichern, ob der Würfelbecher momentan angezeigt wird oder nicht. Die erzeugten Variablen initialisieren wir ebenfalls sinnvoll in der `onCreate`-Methode.
Ergänze die Klasse `MainActivity` entsprechend! Überlege dir dazu, welcher Datentyp sich für die jeweiligen Variablen eignet und welche Werte du zum Initialisieren verwenden möchtest.

8.5 Auslesen des Beschleunigungssensors

Anders als bei den vorherigen Aufgaben müssen wir hier die Daten des Beschleunigungssensors des Mobile Device auslesen, um auf Schütteln reagieren zu können. Anders als beim AppInventor gibt es hier keine konkrete Methode `shaking`, die bei Schütteln unseres Smartphones oder Tablets automatisch aufgerufen wird. Wir müssen uns selber darum kümmern, das Schütteln zu detektieren und von kleineren Änderungen des Beschleunigungssensors (z.B. wenn das Handy vom Tisch aufgenommen oder auf den Tisch abgelegt wird) unterscheiden zu können.

8.5.1 Grundgerüst

Java-Android biete hier ein gewisses Grundgerüst: Jede Activity-Klasse, die auf Daten eines Sensors zugreifen möchte, muss das Interface `SensorEventListener` implementieren. Dazu müssen wir die bisherige Klassendefinition

Code 8.4: Bisherige Klassendefinition

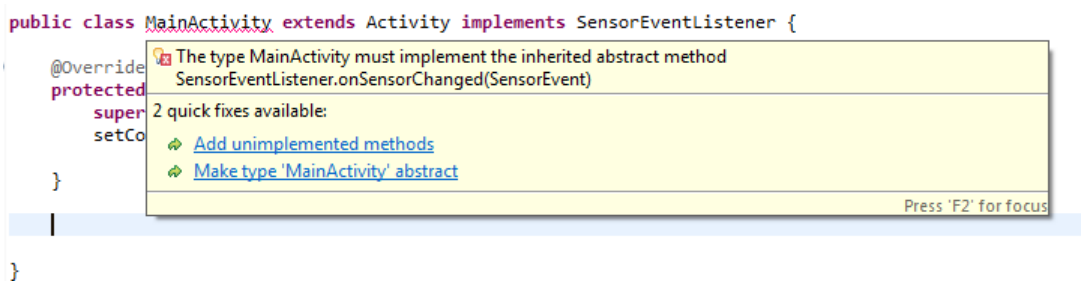
```
public class MainActivity extends ActionBarActivity {
    ...
}
```

um `implements SensorEventListener` erweitern:

Code 8.5: Klasse muss das Interface `SensorEventListener` implementieren.

```
1 public class MainActivity extends ActionBarActivity implements SensorEventListener {
2     ...
3 }
```

Eclipse unterringelt dann automatisch den Klassennamen rot und schlägt uns vor, die nicht-implementierten Methoden automatisch hinzuzufügen:



und erstellt die folgenden beiden Methoden:

Code 8.6: Methoden des Interface `SensorEventListener`

```
1 @Override
2 public void onAccuracyChanged(Sensor sensor, int accuracy) {
3     // TODO Auto-generated method stub
4 }
5
6 @Override
7 public void onSensorChanged(SensorEvent event) {
8     // TODO Auto-generated method stub
9 }
```

8.5.2 Beschleunigungssensor

Bisher haben wir lediglich festgelegt, dass unsere Activity-Klasse auf die Daten eines Sensors zugreifen möchte. Da ein Mobile Device im Normalfall mehr als einen Sensor hat, müssen wir nun festlegen, von welchem Sensor wir Daten empfangen möchten.
Dazu erstellen wir zwei Variablen `SensorManager` und `Sensor`, die wir in der `onCreate`-Methode initialisieren:

Code 8.7: Anmelden des Beschleunigungssensors beim Sensor-Manager

```
1  // Variablen fuer SensorManager und Sensor
2  private SensorManager sensorManager;
3  private Sensor sensorAccelerometer;
4
5  @Override
6  protected void onCreate(Bundle savedInstanceState) {
7      ...
8      // Initialisieren des SensorManagers
9      sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
10     // Initialisieren des Sensors (hier: Beschleunigungssensor)
11     sensorAccelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
12     // Beschleunigungssensor wird beim SensorManager angemeldet
13     sensorManager.registerListener(this, sensorAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
14 }
```

Wir haben nun den Beschleunigungssensor bei unserem Sensor-Manager angemeldet, d.h. ab diesem Zeitpunkt können wir die Daten des Beschleuniungssensors auslesen. Nun kommen aber auch noch die beiden automatisch generierten Methode `onSensorChanged` und `onAccuracyChanged` ins Spiel, wobei wir hier nur die erst genannte verwenden werden. Diese Methode wird immer dann aufgerufen, falls sich die Werte der beim Sensor-Manager angemeldeten Sensoren verändern. Da wir beim Sensor-Manager auch mehr als einen Sensor anmelden können, empfiehlt es sich, in der Methode `onSensorChanged` immer zu überprüfen, welcher der Sensoren den Aufruf der Methode verursacht hat.
Wir erstellen einen neue Methode `wuerfeln`, in der wir anschließend das „Würfeln“ realisieren werden und rufen diese auf, falls der Beschleunigungssensor für das Aufrufen der Methode `onSensorChanged` verantwortlich war:

Code 8.8: Die Methode `onSensorChanged`

```
@Override
public void onSensorChanged(SensorEvent event) {
    //event.sensor gibt den Sensor zurueck, der fuer den Aufruf der Methode verantwortlich war
    Sensor mySensor = event.sensor;

    // Wurde das SensorEvent vom Beschleunigungssensor ausgeloest?
    if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER) {
    wuerfeln(event);
    }
}

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

/**
 * Private Methode, in der das Wuerfeln realisiert wird
 *
 * @param event
 */
private void wuerfeln(SensorEvent event) {
}
```

8.5.3 Die Methode `wuerfeln`

Bevor wir uns um das eigentliche Würfeln kümmern können, müssen wir nun erst einmal entscheiden, ob überhaupt gewürfelt werden soll, d.h. ob das Mobile Device überhaupt geschüttelt wurde. Der Übergabeparameter `SensorEvent` erlaubt es uns, die aktuellen x-, y- und z-Werte des Beschleunigungssensor auszulesen.
Wir verwenden folgende Ideen, um herausfinden zu können, ob geschüttelt worden ist oder nicht:

- zwischen zwei Messungen sollen mindestens 100 Millisekunden vergehen. Wir filtern somit Messwerte heraus - ansonsten würde unsere Berechnung zu oft durchgeführt.
- um feststellen zu können, ob 100 Millisekunden seit der letzten Messung vergangen sind müssen wir uns die Zeit der letzten Messung zwischenspeichern. Wir erstellen dazu eine Variable `lastUpdate` vom Datentyp `long`
- mit der Methode `System.currentTimeMillies()` kann man die aktuelle Zeit auslesen (Rückgabewert ist `long`)
- wir speichern die x-, y- und z-Werte des Beschleunigungssensors der letzten Messung. Dazu erstellen wir entsprechende Variablen vom Datentyp `float`

- wir berechnen aus den gegebenen Werten eine Art Richtwert für die „Beschleunigung“ wie folgt:

$$\texttt{threshold} = \left| \frac{x + y + z - (x^* + y^* + z^*)}{\text{difftime}} \right| \cdot 1000$$

  Dabei sind x, y und z die Werte der aktuellen und x^*, y^* und z^* die Werte der letzten Messung des Beschleunigungssensor sowie `difftime` die Zeit zwischen der aktuellen und der vorherigen Messung

- übersteigt der Wert von `threshold` einen gewissen Schwellwert, soll gewürfelt werden

- als geeigneter Schwellwert wurde durch Experimentieren 45 ermittelt. Wir erstellen dazu eine Konstante

Und nun das ganze in Java-Code:

Code 8.9: Die Methode wuerfeln

```
1  // Variable fuer die zuletzt gemessenen x,y und z-Werte
2  private float last_x, last_y, last_z;
3
4  // Variable fuer die Zeit der letzten Messung
5  private long lastUpate = 0;
6
7  // Schwelle, wann Schuetteln detektiert werden soll
8  private static final int SHAKE_THRESHOLD = 45;
9
10 /**
11  * Private Methode, in der das Wuerfeln realisiert wird
12  *
13  * @param event
14  */
15 private void wuerfeln(SensorEvent event) {
16     // Auslesen der gemessenen Werte fuer x,y und z
17     float[] values = event.values;
18     float x = values[0];
19     float y = values[1];
20     float z = values[2];
21
22     // Messen der aktuellen zeit
23     long curTime = System.currentTimeMillis();
24
25     // Sind seit der letzten Messung mindestens 100 Millisekunden vergangen?
26     // Herausfiltern von Messwerten, da sonst staendig aufgerufen
27     if ((curTime - lastUpate) > 100) {
28         // Berechnen der Zeitspanne, die seit der letzten Messung vergangen ist
29     long diffTime = curTime - lastUpate;
30         // Aktualisieren des Zeitpunktes der letzten Messung
31         lastUpate = curTime;
32         // Wert, um Beschleunigung detektieren zu koennen
33         float threshold = Math.abs(x + y + z - last_x - last_y - last_z) / diffTime * 1000;
34         // Wurde schwelle ueberschritten, d.h. Schuetteln detektiert?
35         if (threshold > SHAKE_THRESHOLD) {
36
37             //TODO: Werte fuer Wuerfel generieren und Wuerfel anzeigen
38
39         }
40     // Aktualisierung der zuletzt gemessenen x,y und z-Werte
41     last_x = x;
42     last_y = y;
43     last_z = z;
44     }
45 }
```

Wir müssen uns nun nur noch um den mit `TODO` gekennzeichneten Bereich kümmern. Hier müssen wir zunächst den Würfelbecher auf sichtbar und die TextView-Objekte für die Würfel auf unsichtbar setzen. Anschließend erzeugen wir zwei Zufallszahlen zwischen 1 und 6. Hier können wir auf die vordefinierte Methode `Math.random()` zurückgreifen, die zunächst eine Zufallzahl zwischen 0 und 1 erzeugt. Diese Zahl multiplizieren wir mit 6. Da der Cast auf einen Integer immer abrundet[2], müssen wir noch plus 1 rechnen. Nun müssen wir nur noch je nach Ziffer das gewünschte Bild als Hintergrund der TextView-Objekte festlegen. Dies geht am besten mit einer `switch`-Anweisung:

Code 8.10: Sichtbarkeit ändern und Würfelergebnisse anzeigen

```
1  // Bei Schuetteln wird der Becher angezeigt
2  if (!becherVisible) {
3      wuerfelbecher.setVisibility(View.VISIBLE);
4      wuerfel1.setVisibility(View.INVISIBLE);
5      wuerfel2.setVisibility(View.INVISIBLE);
6      becherVisible = true;
7  }
8
9  // Wuerfelwurf
10 ergebnis1 = (int) (Math.random() * 6) + 1;
11 ergebnis2 = (int) (Math.random() * 6) + 1;
12
13 // Je nach Wuerfelergebnis wird das entsprechende
14 // Hintergrundbild gesetzt
15 switch (ergebnis1) {
16     case 1:
17     wuerfel1.setBackgroundResource(R.drawable.eins);
18     break;
19
20     case 2:
21     wuerfel1.setBackgroundResource(R.drawable.zwei);
22     break;
23
24     case 3:
25     wuerfel1.setBackgroundResource(R.drawable.drei);
26     break;
27
28     case 4:
29     wuerfel1.setBackgroundResource(R.drawable.vier);
30     break;
31
32     case 5:
33     wuerfel1.setBackgroundResource(R.drawable.fuenf);
34     break;
35
36     case 6:
37     wuerfel1.setBackgroundResource(R.drawable.sechs);
38     break;
39
40     default:
41     break;
42 }
43
44 // Je nach Wuerfelergebnis wird das entsprechende
45 // Hintergrundbild gesetzt
46 switch (ergebnis2) {
47     case 1:
48     wuerfel2.setBackgroundResource(R.drawable.eins);
49     break;
50
51     case 2:
```

[2] `(int) 99.999` liefert den Wert `99`.

```
52      wuerfel2.setBackgroundResource(R.drawable.zwei);
53      break;
54
55      case 3:
56      wuerfel2.setBackgroundResource(R.drawable.drei);
57      break;
58
59      case 4:
60      wuerfel2.setBackgroundResource(R.drawable.vier);
61      break;
62
63      case 5:
64      wuerfel2.setBackgroundResource(R.drawable.fuenf);
65      break;
66
67      case 6:
68      wuerfel2.setBackgroundResource(R.drawable.sechs);
69      break;
70
71      default:
72      break;
73  }
```

8.6 Würfelbecher

Wir sind nun fast fertig. Das einzige was noch fehlt ist, dass wir bei Klicken auf den Würfelbecher den Becher nicht mehr sehen sollen, sondern die Würfel. Dazu erstellen eine Methode `onBecherClick`, die bei Klicken auf den Würfelbecher aufgerufen werden soll. Vergessen Sie also nicht, dies in der Layout-Datei entsprechend festzulegen[3].

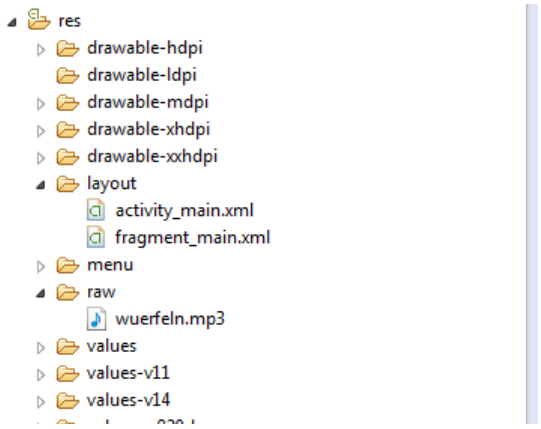
Code 8.11: Die Methode `onBecherClick`

```
1   /**
2    * Wird aufgerufen, wenn auf den Wuerfelbecher geklickt wird
3    *
4    * @param view
5    */
6   public void onBecherClick(View view) {
7       // Aendert die Sichtbarkeit von Becher und "Wuerfeln"
8       wuerfelbecher.setVisibility(View.INVISIBLE);
9       wuerfel1.setVisibility(View.VISIBLE);
10      wuerfel2.setVisibility(View.VISIBLE);
11      becherVisible = false;
12  }
```

[3]Siehe ggf. auf Seite **??** nach, wie das geht.

8.7 Würfelgeräusch abspielen

Wie beim AppInventor können wir natürlich auch hier ein Würfelgeräusch ausgeben, wenn das Mobile Device geschüttelt wird. Dazu legen wir die entsprechende Musik-Datei `wuerfeln.mp3` in `res/raw` ab:



Die Datei können wir nun an geeigneter Stelle mit folgenden Befehlen abspielen:

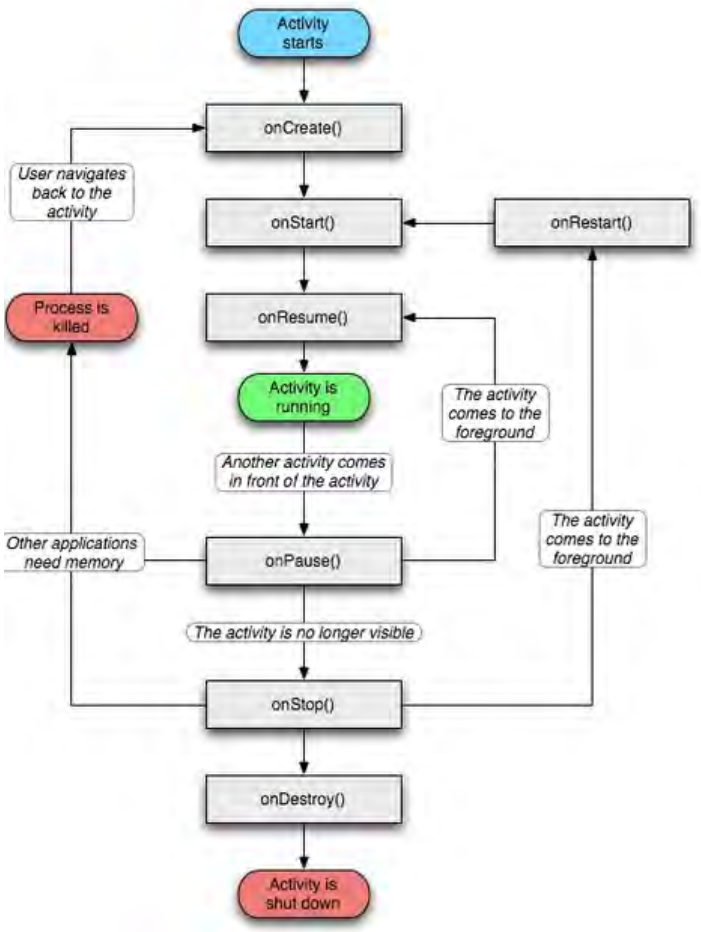
Code 8.12: MediaPlayer zum Abspielen des Würfel-Geräusches

```
1 MediaPlayer mp = MediaPlayer.create(getApplicationContext(), R.raw.wuerfeln);
2 mp.start();
```

8.8 Ressourcen sparen

Wir haben beim Sensor-Manager den Beschleunigungssensor angemeldet. Dieser ruft wie bereits erwähnt nun ständig bei Änderung der Werte die Methode `onSensorChanged` auf, auch wenn die Applikation im Hintergrund läuft. Dies geht natürlich auf Kosten der Akku-Laufzeit. Aus diesem Grund wäre es sinnvoll, dass uns die Werte des Beschleunigungssensors nur interessieren, wenn die App auch wirklich verwendet wird.

Werfen wir hierzu einen Blick auf den „Lebenszyklus“ einer Android-Applikation:



Es wäre also gut, wenn wir bei `onPause()` den Beschleunigungssensor beim Sensor-Manager abmelden und ihn erst wieder anmelden, wenn die Applikation wieder in den Fokus (in den Vordergrund tritt), also bei Aufruf der Methode `onResume()`. Dazu überschreiben wir die beiden vordefinierten Methoden wie folgt:

Code 8.13: MediaPlayer zum Abspielen des Würfel-Geräusches

```
1  @Override
2  protected void onResume() {
3      super.onResume();
4      sensorManager.registerListener(this, sensorAccelerometer,SensorManager.SENSOR_DELAY_NORMAL);
5  }
6
7  @Override
8  protected void onPause() {
9      super.onPause();
10     sensorManager.unregisterListener(this);
11 }
```

Kapitel 9

Datenspeicher

Überblick:

In diesem Kapitel werden wir die Datenspeicher-Aufgabe aus dem AppInventor-Modul in Java Android programmieren.

Wir möchten Ihnen dabei anhand einer zweiten Aufgabe die Möglichkeit bieten, Vergleiche zwischen der App-Programmierung mit dem AppInventor und der App-Programmierung mit Java-Android ziehen zu können.

Welche Vor- bzw. Nachteile hat das jeweilige Werkzeug?

9.1 Aufgabenstellung

Wir möchten eine Applikation erstellen, bei der man den Vor- und Nachnamen, das Geburtsdatum und die Email-Adresse abspeichern und bei Bedarf alle abgespeicherten Kontakt anzeigen kann.

9.2 Projekt anlegen

Zunächst müssen wir ein Android-Applikation-Projekt anlegen. Wir gehen hier genauso vor, wie in Kapitel 4 in Abschnitt 4.1. Als App-Namen verwenden wir *Datenspeicher* und als App-Symbol ein geeignetes Bild:

9.3 MainActivity und Layout

Zunächst ändern wir die Klasse `MainActivity.java`[1] leicht ab, da wir einige generierte Einstellungen nicht benötigen:

Code 9.1: MainActivity.java

```
1  package com.example.datenspeicher;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class MainActivity extends Activity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

Nun können wir das Layout festlegen, das beim Starten der Applikation gezeigt werden soll. Das Layout sollte etwa wie auf Seite 90 aussehen. Wir haben zum Positionieren der verschiedenen View-Objekte ein `RelativeLayout` verwendet. Zur Darstellung der Texte *Kontaktformular, Name, Vorname, Geburtsdatum, E-Mail-Adresse* haben wir TextView-Objekte verwendet, für die entsprechenden Eingabeformulare EditText-Objekte mit geeigneten Hints (z.B. `TT.MM.JJJJ`). Hierzu fügt man in der XML-Ansicht bei den EditText-Objekten die Zeile

```
android:hint=" .. text .. "
```

[1] `Datenspeicher` → `src` → `com.example.datenspeicher`

hinzu. Den beiden Button-Objekten *Kontakt abspeichern* und *Kontakte anzeigen* haben wir ein Hintergrundbild zugewiesen:

9.4 Die Klasse DataHandler

Um die Kontaktdaten persistent speichern zu können, benötigen wir eine appinterne Datenbank. Den Zugriff auf diese verwalten wir wie in Kapitel 5 mit einer zusätzlichen Klasse - wir nennen diese `DataHandler`.

9.4.1 Grundgerüst

Die `DataHandler`-Klasse hat folgenden Aufbau[2]

Code 9.2: DataHandler.java

```
1  package com.example.datenspeicher;
2
3  import android.content.Context;
4  import android.database.sqlite.SQLiteDatabase;
5  import android.database.sqlite.SQLiteOpenHelper;
6
7  public class DataHandler {
8
9      private static final String DATABASE_NAME = "kontaktDB";
10     private static final int DATABASE_VERSION = 1;
11
12     private DataBaseHelper dbHelper;
13     private Context ctx;
14     private SQLiteDatabase db;
15
16     public DataHandler(Context ctx) {
17         this.ctx = ctx;
18         dbHelper = new DataBaseHelper(this.ctx);
19     }
20
21
22     /**
23      * Private innere Klasse DataBaseHelper
24      *
25      * @author Administrator
26      *
27      */
28     private static class DataBaseHelper extends SQLiteOpenHelper {
29
30         public DataBaseHelper(Context ctx) {
31             super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
32         }
33
34         @Override
35         public void onCreate(SQLiteDatabase db) {
36             db.execSQL("create table if not exists kontaktTable(name text not null, " + "vorname text not
                   null, gebdatum text not null, mail text not null);");
37         }
38
39         @Override
40         public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
41             // TODO Auto-generated method stub
```

[2] für eine ausführliche Beschreibung des Aufbaus siehe Kapitel 5 Abschnitt 5.10.

```
42          }
43      }
44  }
```

In der inneren Klasse `DataBaseHelper`-Klasse erstellen wir eine Tabelle mit den Spalten *name, vorname, gebdatum* und *mail*. Das entsprechende SQLite-Statement

```
CREATE TABLE IF NOT EXISTS kontaktTable(
    name        text NOT NULL,
    vorname     text NOT NULL,
    gebdatum    text NOT NULL,
    mail        text NOT NULL);
```

können wir mit dem Java-Befehl `execSQL(...)` wie gewohnt direkt im Java-Code ausführen.

9.4.2 Methoden für Lese- und Schreibzugriff

Um aus unserer Activity-Klasse lesend bzw. schreibend auf die Datenbank zugreifen zu können, benötigen wir in der `DataHandler`-Klasse noch entsprechende Methoden. Wir nennen diese `openWrite()` und `openRead()`. Nach dem Zugriff sollen die Rechte entzogen werden. Dazu erstellen wir die Methode `close()`:

Code 9.3: Methoden für Lese- und Schreibzugriff

```
1  /**
2   * Speichert in der Variable db unsere Datenbank mit Schreib-Rechten
3   * @return
4   */
5  public void openWrite() {
6      db = dbHelper.getWritableDatabase();
7  }
8
9  /**
10  * Speichert in der Variable db unsere Datenbank mit Lese-Rechten
11  * @return
12  */
13 public void openRead() {
14     db = dbHelper.getReadableDatabase();
15 }
16
17 /**
18  * Nach Ausfuehrung von Lese- oder Schreibzugriffen werden die Zugriffsrechte auf die Datenbank wieder
        zurueckgesetzt
19  */
20 public void close() {
21     dbHelper.close();
22 }
```

9.5 View-Objekte in der MainActivity-Klasse verwalten

Wir erstellen für die in der Layout-Datei verwendeten EditText-Objekte Variablen in der MainActivity-Klasse und initialisieren diese in der `onCreate`-Methode. Die View-Objekte „finden“ wir mit der Methode `findViewById` über Ihre ID, die wir ihnen in der Layout-Datei zugewiesen haben. Weiter erstellen wir ein DataHandler-Objekt für den Datenbankzugriff:

Code 9.4: Methoden für Lese- und Schreibzugriff

```
/*
 * Variablen fuer die View-Objekte
 */
private EditText editName;
private EditText editVorname;
private EditText editGebDatum;
private EditText editMail;
private Spinner spinner;

// DataHandler-Objekt fuer den Datenbankzugriff
private DataHandler dbHandler;

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

    // Initalisieren des DataHandler-Objekts fuer den Datenbankzugriff
    dbHandler = new DataHandler(this);

    // Initialisieren der View-Objekte
    editName = (EditText) findViewById(R.id.editName);
    editVorname = (EditText) findViewById(R.id.editVorname);
    editGebDatum = (EditText) findViewById(R.id.editGeb);
    editMail = (EditText) findViewById(R.id.editMail);
}
```

9.6 Neuen Kontakt hinzufügen

Um einen neuen Kontakt hinzufügen zu können, benötigen wir zunächst eine Methode in der `MainActivity`-Klasse, die aufgerufen wird, wenn auf den entsprechenden Button geklickt wird. Was muss nun alles gemacht werden?
Erstens soll überprüft werden, ob auch alle Eingabefelder ausgefüllt worden sind, d.h. nicht leer sind. Anschließend möchten wir die eingegebenen Werte in der Tabelle `kontaktTable` unserer internen Datenbank speichern. Dazu benötigen wir noch eine Methode in der `DataHandler`-Klasse mit einem geeigneten SQL-Befehl. Fertig ☺

9.6.1 Methode `addKontakt` in der `DataHandler`-Klasse

Kümmern wir uns zunächst um die entsprechende Methode in der `DataHandler`-Klasse. Unsere Methode braucht vier Übergabeparameter, d.h. folgende Methodendefinition:

Code 9.5: Methodendefinition von `addKontakt`

```
/**
 * Neuen Kontakteintrag in Datenbank machen
 *
 * @param name
 * @param vorname
 * @param gebDatum
 * @param mail
 */
public void addKontakt(String name, String vorname, String gebDatum, String mail) {

}
```

Nun benötigen wir nur noch einen geeigneten SQL-Befehl, um einen neuen Eintrag in unsere Tabelle `kontaktTable` einfügen zu können:

```
INSERT INTO kontaktTable(name,vorname,gebDatum,mail)
    VALUES('neuer_name','neuer_vorname','neues_gebdatum','neue_mail');
```

Diesen führen wir in der Methode aus. Wichtig ist hier, dass die einfachen Anführungszeichen des SQL-Befehls nicht vergessen werden. Mit der String-Syntax von Java kann es schnell passieren, dass hier ein Syntax-Fehler im SQL-Befehl entsteht:

Code 9.6: Die Methode `addKontakt`

```
/**
 * Neuen Kontakteintrag in Datenbank machen
 *
 * @param name
 * @param vorname
 * @param gebDatum
 * @param mail
 */
public void addKontakt(String name, String vorname, String gebDatum, String mail) {
      db.execSQL("insert into kontaktTable(name,vorname,gebDatum,mail) values('" + name + "', '"
            + vorname + "','" + gebDatum + "','" + mail + "');");
}
```

9.6.2 Die Methode `saveButtonClicked` in der Klasse `MainActivity`

Die Methode `saveButtonClicked` soll aufgerufen werden, falls auf die Schaltfläche *Kontakt* speichern geklickt wird. Ergänzen Sie dazu eine geeignete Zeile in der XML-Ansicht der zugehörigen Layout-Datei. Vergessen Sie auch nicht die dafür benötigte Methodensyntax mit Aufrufparameter der Klasse `View`.
Bei Methodenaufruf müssen wir nun zunächst die eingegebenen Daten aus den Textfeldern auslesen. Anschließend überprüfen wir, ob auch alle Kontaktfelder ausgefüllt wurden. Ist dies der Fall, nutzen wir die in Abschnitt 9.6.1 geschriebene Methode der `DataHandler`-Klasse zum abspeichern des Kontaktes:

Code 9.7: Die Methode `saveButtonClicked`

```
/**
 * Wird aufgerufen, wenn auf die Speichern-Schaltflaeche geklickt wird
 * Speichert die eingegebenen Kontaktdaten als neuen Kontakt
 * @param view
 */
public void saveButtonClicked(View view) {
    // Auslesen der eingegebenen Kontaktdaten
    String name = editName.getText().toString();
    String vorname = editVorname.getText().toString();
    String gebDatum = editGebDatum.getText().toString();
    String mail = editMail.getText().toString();

    // Falls ein Feld leer war, wird der Benutzer benachrichtigt
    if (name.equals("") || vorname.equals("") || gebDatum.equals("") || mail.equals("")) {
        Toast.makeText(this, "Bite alle Felder ausfuellen", Toast.LENGTH_LONG).show();
    // ansonsten wird ein neuer Kontakt in der Datenbank abgespeichert
    } else {
        dbHandler.openWrite();
        dbHandler.addKontakt(name, vorname, gebDatum, mail);
        dbHandler.close();
        Toast.makeText(this, "Kontakt hinzugefuegt", Toast.LENGTH_LONG).show();
  }
}
```

9.7 Kontakte anzeigen

Wenn auf die Schaltfläche *Kontakte anzeigen* geklickt wird, soll eine neue Layout-Datei angezeigt werden, in der alle gespeicherten Kontakte in einer Liste (`Spinner`) angezeigt werden. Wir benötigen somit:

- eine neue Layout-Datei für das Anzeigen der gespeicherten Kontakte
- eine Methode `getKontakte` in der Klasse `DataHandler`, die alle Einträge aus der Tabelle `kontaktTable` holt und zurückgibt
- eine Methode `showButtonClicked` in der Klasse `MainActivity`, die aufgerufen wird, wenn auf die Schaltfläche *Kontakt anzeigen* geklickt wird. In dieser Methode müssen wir
  - mittels der Methode `getKontakte` die Einträge aus der Datenbank auslesen
  - die Layout-Datei ändern
  - die Liste mit den ausgelesenen Kontakten füllen

9.7.1 Layout-Datei `activity_kontakte.xml`

Wir erstellen in dem Ordner `res/layout` eine neue Layout-Datei `activity_kontakte.xml`[3]. In dieser erstellen wir ein TextView-Objekt für die Überschrift, ein Spinner-Objekt für die Kontaktliste und zwei Button-Objekte *Kontakt löschen* und *Kontakte hinzufügen*.

Um die Schaltfläche *Kontakt löschen* kümmern wir uns später. Mit der Schaltfläche *Kontakte hinzufügen* sollen wir zu der anderen Layout-Datei zurückspringen können.

Die Layout-Datei sollte etwa wie folgt aussehen:

9.7.2 Die Methode `getKontakte` in der Klasse `DataHandler`

Als nächstes kümmern wir uns um die Methode `getKontakte` in der `DataHandler`-Klasse. Da wir das Spinner-Objekt mit einer `ArrayList<String>` „füllen“ können, wählen wir als Rückgabetyp der Methode den Datentyp `ArrayList<String>`:

[3] Rechtsklick auf den Ordner `layout` und dann `New` → `Android XML File` .

Code 9.8: `ArrayList<String>` als Rückgabewert der Methode `getKontakte`

```
/**
 * Gibt alle Eintrag als ArrayList zurueck
 *
 * @return
 */
public ArrayList<String> getKontakte() {

}
```

Nun brauchen wir ein SQL-Statement, das uns alle Zeilen der Tabelle `kontaktTable` liefert. Gleichzeitig sollen die Einträge zunächst nach dem Nachnamen und in zweiter Ebene nach dem Vornamen sortiert werden:

```
SELECT name, vorname, gebdatum, mail
FROM kontaktTable
ORDER BY name, vorname
```

Diese SQL-Anfrage können wir mit dem Befehl `db.rawQuery( .. )` an die Datenbank stellen und erhalten als Rückgabewert ein Cursor-Objekt. Die Funktionsweise des Cursor-Objekts haben wir bereits in Abschnitt ??? ausführlich dargestellt und gehen an dieser Stelle nicht mehr genauer darauf ein.
Wir erstellen uns am Anfang der Methode eine `ArrayList<String> kontaktliste`, die wir nun mit den im Cursor gespeicherten Werten befüllen. Jeder Eintrag, also Name, Vorname, Geburtsdatum und E-Mail-Adresse, wird nun zu **einem** String zusammengefügt. Zur Übersichtlichkeit fügen wir nach Name, Vorname und Geburtsdatum jeweils einen Zeilenumbruch (`\n`) ein:

Code 9.9: Die Methode `getKontakte`

```
/**
 * Gibt alle Eintrag als ArrayList zurueck
 *
 * @return
 */
public ArrayList<String> getKontakte() {
    ArrayList<String> kontaktliste = new ArrayList<String>();
    String eintrag;
    Cursor c = db.rawQuery(
    "select * from kontaktTable order by name,vorname", null);

    // Wurde mindestens ein Kontakt in der Datenbank gefunden?
    if (c.moveToFirst()) {
        eintrag = c.getString(0) + " \n" + c.getString(1) + " \n" + c.getString(2) + " \n" + c.getString(3);
        kontaktliste.add(eintrag);
        // Alle weiteren Kontakte im Cursor werden durchlaufen
        while (c.moveToNext()) {
            eintrag = c.getString(0) + " \n" + c.getString(1) + " \n" + c.getString(2) + " \n" +
                c.getString(3);
            kontaktliste.add(eintrag);
        }
    }
    return kontaktliste;
}
```

9.7.3 Die Methode `showButtonClicked` in der Klasse `MainActivity`

Die Methode `showButtonClicked` soll aufgerufen werden, falls auf die Schaltfläche *Kontakte anzeigen* geklickt wird. Ergänzen Sie dazu eine geeignete Zeile in der XML-Ansicht der zugehörigen Layout-Datei. Vergessen Sie

auch nicht die dafür benötigte Methodensyntax mit Aufrufparameter der Klasse `View`.

Bei Methodenaufruf müssen wir nun zunächst die gespeicherten Kontakte aus der Datenbank auslesen - hierfür haben wir bereits in Abschnitt 9.7.2 die Methode `getKontakte` in der Klasse `DataHandler` erstellt. Anschließend müssen wir die Layout-Datei wechseln. Dies geht mit dem Befehl `setContentView( .. )`.

Nun müssen wir dem Spinner nur noch die Elemente unserer `ArrayList` zuweisen. Dies geht über ein Objekt der Klasse `ArrayAdaper`[4]:

Code 9.10: Die Methode `showButtonClicked`

```
/**
* Wird aufgerufen, wenn auf die Alle Kontakte anzeigen Schaltflaeche
* geklickt wird Zeigt alle Kontakte an, die sich in der Datenbank befinden
*
* @param view
*/
public void showButtonClicked(View view) {
    // Holt alle Kontakte aus der Datenbank
    dbHandler.openRead();
    ArrayList<String> kontaktListe = dbHandler.getKontakte();
    dbHandler.close();

    // Sind Kontakte vorhanden, wird das Layout geaendert und die Kontakte
    // im Spinner angezeigt
    if (kontaktListe.size() > 0) {
        setContentView(R.layout.activity_kontakte);
        spinner = (Spinner) findViewById(R.id.kontaktSpinner);
        ArrayAdapter<String> spinnerAdapter = new ArrayAdapter<String>(this, R.layout.spinnerlayout,
            kontaktListe);
        spinnerAdapter.setDropDownViewResource(R.layout.spinnerlayout);
        spinner.setAdapter(spinnerAdapter);
    } else {
        Toast.makeText(this, "Keine Kontakte gespeichert", Toast.LENGTH_LONG).show();
    }
}
```

9.7.4 Die Methode `backButtonClicked` in der Klasse `MainActivity`

In der Layout-Datei `activity_kontakte.xml` haben wir zwei Schaltflächen erstellt. Über die Schaltfläche *Kontakte hinzufügen*, soll die Layout-Datei `activity_main.xml` angezeigt werden. Hierfür erstellen wir eine Methode `backButtonClicked` und legen fest, dass diese bei Klicken auf die Schaltfläche *Kontakte hinzufügen* aufgerufen wird.
In der Methode müssen wir nur noch die Layout-Datei wechseln:

Code 9.11: Die Methode `showButtonClicked`

```
public void backButtonClicked(View view) {
    setContentView(R.layout.activity_main);
}
```

[4] vgl. auch Seite ??.

Nun können wir uns die gespeicherten Kontakte anzeigen lassen:

9.8 Kontakte löschen

Abschließend soll es möglich sein, gespeicherte Kontakte auch wieder löschen zu können.

Um einen Eintrag löschen zu können, benötigen wir natürlich die Informationen zu Name, Vorname, Geburtsdatum und E-Mail-Adresse. Aus der Liste mit den gespeicherten Kontakten können wir diese Informationen leider nicht mehr herausholen, da für jeden Kontakt alle vier Daten zu **einem** String zusammengefügt wurden. Aus diesem Grund müssen wir uns eine andere Möglichkeit überlegen, um an die Daten zu kommen.

Es bietet sich dabei folgende Idee an: die Kontakte sind nach Name und Vorname sortiert in der Spinner-Liste. Zu jedem Kontakt in der Spinner-Liste können wir dessen Position abfragen. Wenn wir nun eine erneute Anfrage an die Datenbank stellen, die alle Kontakte nach Name und Vorname sortiert aus der Datenbank ausliest und in einem Cursor-Objekt zurückgibt, können wir anhand der Position des Kontaktes, die Einzelinformationen aus dem Cursor auslesen.

Beispiel:

Wir wählen etwa den vierten Kontakt in der Liste aus und drücken die Schaltfläche *Kontakt löschen*. Wir wissen also, dass wir, wenn wir alle Kontakte nach Name und Vorname sortieren, den vierten Kontakt löschen wollen. Also stehen an der vierten Position im Cursor-Objekt die Einzeldaten des Kontaktes, den wir löschen wollen. Mit den Einzeldaten können wir mittels SQL-Befehl den Eintrag aus der Datenbank löschen.

Wir benötigen also:

- eine Methode `getKontaktByPos(int pos)` in der `DataHandler`-Klasse, die einen Kontakt anhand seiner Position aus der Datenbank ausliest

- eine Methode `deleteKontakt(String[] kontakt)` in der `DataHandler`-Klasse, die einen Kontakt aus der Datenbank löscht
- eine Methode `deleteButtonClicked(View view)` in der `MainActivity`-Klasse, die aufgerufen wird, wenn auf die Schaltfläche *Kontakt löschen* geklickt wird. In dieser Methode müssen wir
  - ermitteln, an welcher Position der aktuell im Spinner ausgewählte Kontakt steht
  - die Kontaktdaten anhand er Position aus der Datenbank holen (über die Methode `getKontaktByPos`)
  - den Kontakt aus der Datenbank löschen (über die Methode `deleteKontakt`)
  - den Spinner aktualisieren

9.8.1 Die Methode `getKontaktByPos` in der `DataHandler`-Klasse

In der Methode `getKontaktByPos` müssen wir die vier Daten eines Kontaktes über seine Position aus der Datenbank holen. Die Position übergeben wir im Aufrufparameter als `int`. Als Rückgabetyp verwenden wir ein String-Array mit vier Einträgen.
Wir holen nun zunächst alle Daten, sortiert nach Name und Vorname, aus der Datenbank und holen dann die gewünschten Daten entsprechend der Position aus dem Cursor-Objekt:

Code 9.12: Die Methode `getKontaktByPos`

```
/**
* Liest einen Kontakt ueber seine Position in der Datenbank aus und gibt
* diesen in einem String-Array zurueck
*
* @param pos
* @return
*/
public String[] getKontaktByPos(int pos) {
    String[] kontakt = new String[4];
    Cursor c = db.rawQuery("select * from kontaktTable order by name,vorname", null);

    if (c.moveToPosition(pos)) {
        kontakt[0] = c.getString(0);
        kontakt[1] = c.getString(1);
        kontakt[2] = c.getString(2);
        kontakt[3] = c.getString(3);
    }
    return kontakt;
}
```

9.8.2 Die Methode `deleteKontakt` in der `DataHandler`-Klasse

In der Methode `deleteKontakt` löschen wir einen Kontakt aus der Datenbank. Die Informationen zu Name, Vorname, Geburtsdatum und E-Mail übergeben wir im Aufrufparameter als String-Array.

Zunächst benötigen wir noch eine SQL-Anfrage, mit der wir einen Kontakt löschen können:

```
DELETE FROM kontaktTable
WHERE name = 'name' AND vorname = 'vorname' AND gebdatum = 'gebdatum'
        AND mail = 'mail'
```

Code 9.13: Die Methode `deleteKontakt`

```
/**
* Loescht den Kontakt mit den Daten, die im String-Array kontakt uebergeben
```

```
3  * werden
4  *
5  * @param kontakt
6  */
7  public void deleteKontakt(String[] kontakt) {
8      db.execSQL("delete from kontaktTable where name='" + kontakt[0] + "' and vorname='" + kontakt[1]
9          + "' and gebDatum='" + kontakt[2] + "' and mail='" + kontakt[3] + "';");
10 }
```

9.8.3 Die Methode `deleteButtonClicked` in der `MainActivity`-Klasse

Die Methode `deleteButtonClicked` soll aufgerufen werden, wenn auf die Schaltfläche *Kontakt löschen* geklickt wird. Ergänzen Sie dazu eine geeignete Zeile in der XML-Ansicht der zugehörigen Layout-Datei. Vergessen Sie auch nicht die dafür benötigte Methodensyntax mit Aufrufparameter der Klasse `View`.

Zunächst ermitteln wir, an welcher Position der aktuell im Spinner ausgewählte Kontakt steht:

Code 9.14: Position des Kontaktes im Spinner ermitteln

```
1 spinner = (Spinner) findViewById(R.id.kontaktSpinner);
2 int pos = spinner.getSelectedItemPosition();
```

Als nächstes ermitteln wir die Kontaktdaten des zu löschenden Kontaktes

Code 9.15: Kontaktdaten aus der Datenbank auslesen

```
1 // Kontakt wird ueber die Position ermittelt
2 dbHandler.openRead();
3 String[] kontakt = dbHandler.getKontaktByPos(pos);
4 dbHandler.close();
```

und löschen diesen anschließend aus der Tabelle `kontaktTable` unserer Datenbank:

Code 9.16: Kontakt aus der Datenbank löschen

```
1 // Ausgewaehlter Kontakt wird aus der Datenbank geloescht
2 dbHandler.openWrite();
3 dbHandler.deleteKontakt(kontakt);
4 dbHandler.close();
```

Abschließend müssen wir nun noch den Spinne aktualisieren, denn bisher wird der gelöschte Kontakt immer noch angezeigt. Dazu müssen wir im Grunde genommen dasselbe machen, wie beim Klicken auf die Schaltfläche *Kontakte anzeigen*, also wie in der Methode `showButtonClicked`.
Wir erstellen hierzu eine private Hilfsmethode `updateSpinner()`, die den Spinner aktualisiert:

Code 9.17: Die Methode `updateSpinner()`

```
1  /**
2  * Aktualisiert die Eintraege im Spinner
3  */
4  private void updateSpinner() {
5      // Auslesen aller Kontakte aus der Datebank
6      dbHandler.openRead();
7      ArrayList<String> kontaktListe = dbHandler.getKontakte();
8      dbHandler.close();
9
10     // Kontakte im Spinner anzeigen
```

```
11      spinner = (Spinner) findViewById(R.id.kontaktSpinner);
12      ArrayAdapter<String> spinnerAdapter = new ArrayAdapter<String>(this, R.layout.spinnerlayout,
            kontaktListe);
13      spinnerAdapter.setDropDownViewResource(R.layout.spinnerlayout);
14      spinner.setAdapter(spinnerAdapter);
15  }
```

Die Methode `deleteButtonClicked` sieht somit wie folgt aus:

Code 9.18: Die Methode `deleteButtonClicked`

```
1   /**
2   * Wird aufgerufen, wenn auf die Loeschen-Schaltflaeche geklickt wird
3   * Loescht den im Spinner ausgewaehlten Kontakt
4   *
5   * @param view
6   */
7   public void deleteButtonClicked(View view) {
8       spinner = (Spinner) findViewById(R.id.kontaktSpinner);
9       int pos = spinner.getSelectedItemPosition();
10
11      // Kontakt wird ueber die Position ermittelt
12      dbHandler.openRead();
13      String[] kontakt = dbHandler.getKontaktByPos(pos);
14      dbHandler.close();
15
16      // Ausgewaehlter Kontakt wird aus der Datenbank geloescht
17      dbHandler.openWrite();
18      dbHandler.deleteKontakt(kontakt);
19      dbHandler.close();
20
21      updateSpinner();
22  }
```

Kapitel 10

LegoPilot-App

Überblick:

In diesem Kapitel möchten wir einen LegoEV3-Roboter via Handy steuern. Die Verbindung zwischen Handy und Roboter werden wir über WLAN herstellen; wir werden hierzu einen einfachen Router ohne Verbindung zum Internet verwenden.
Da uns die Lego-Software nicht ermöglicht, eine Netzwerkverbindung zu einem anderen Gerät herzustellen, werden wir LEJOS verwenden, um den Roboter zu programmieren. Die Installation und Programmierung des EV3-Roboters mit LEJOS behandeln wir in dem Modul Roboter-Programmierung und verweisen hier auf die entsprechenden Kapitel darin.

10.1 Aufgabenstellung und Inhaltliche Voraussetzungen

Für die Kommunikation brauchen wir zwei Programme. Eine App, die in diesem Fall als Server dient und je nach Benutzereingabe die entsprechenden Befehle über die Netzwerkverbindung an den Roboter schickt und ein Programm auf dem Roboter, das auf entsprechende Befehle des Smartphones wartet und entsprechend dieser die Motoren des Roboters ansteuert.

Für diese Aufgabe werden grundlegende Java-Befehle vorausgesetzt. Zudem sollten Sie bereits folgende Aufgaben bearbeitet haben:

- Client-Server-Kommunikation
- Aufgaben mit LEJOS

10.2 Wichtiges zur Netzwerkverbindung

Wie man Anfang dieses Kapitels bereits erwähnt, benötigen wir für die Netzwerkverbindung WLAN. Sowohl Handy als auch EV3 Roboter benötigen eine IP-Adresse im gleichen WLAN-Netz. Wie Sie dem EV3-Roboter in einem WLAN-Netzwerk anmelden, können Sie ebenfalls in dem Modul Roboter-Programmierung nachlesen. Wir verwenden zwischen beiden Geräten eine Socket-Verbindung (TCP) aufbauen, wobei wir diese nur uni- und nicht bidirektional nutzen werden, d.h. es werden nur Befehle vom Handy zum Roboter geschickt und nicht umgekehrt.
Die Art der Kommunikation haben wir in Abbildung 10.1 visualisiert.

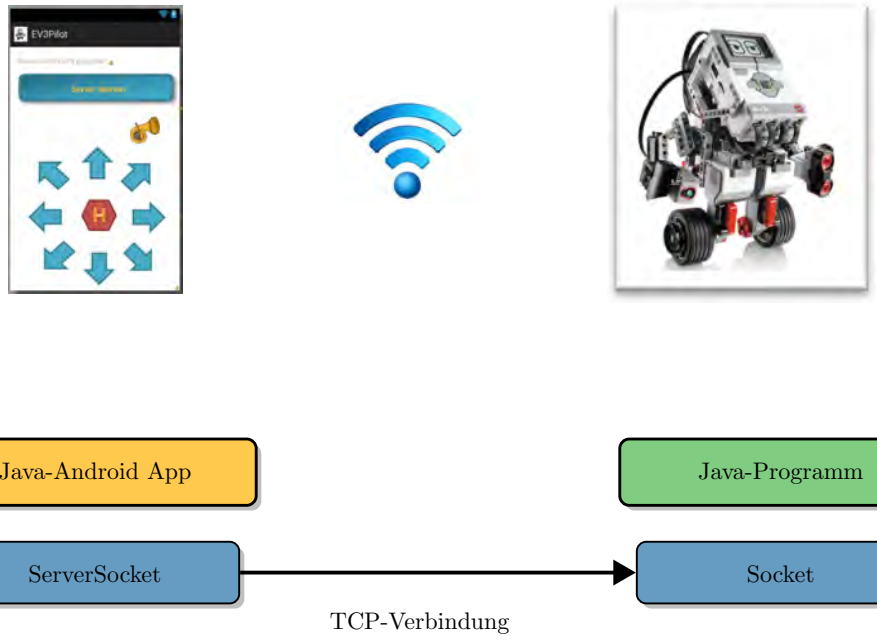


Abbildung 10.1: App und Roboter kommunizieren über eine Socket-Verbindung im gemeinsamen WLAN

10.2.1 ServerSocket und Socket

Aus der Aufgabe in Kapitel **??** wissen Sie bereits, dass im Java-Programm des Servers ein `ServerSocket` angelegt wird, der auf die Anmeldung eines Clients wartet.
Im Gegensatz zu unserem Server im vorangegangenen Kapitel müssen wir beim Anlegen des `ServerSocket` bei Java-Android zwei wichtige Sachen beachten:

(1) **Erlaubnis für Netzwerkverbindung setzen**
Die von uns erstellte Applikation hat standardmäßig keine Erlaubnis um auf Netzwerkverbindungen herzustellen. Wir müssen ihr diese Erlaubnis explizit erteilen. Dies erreichen wir, indem wir in der XML-Datei `AndroidManifest.xml` folgende Zeile einfügen:

Code 10.1: Der Applikation muss die Erlaubnis für das Erstellen von Netzwerkverbindungen erteilt werden.

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

(2) **Netzwerkverbindung nicht auf UIThread**
Java-Android lässt nicht zu, einen `ServerSocket` auf dem `UIThread` zu erstellen. `UIThread` steht für *User-Interface-Thread* und bezeichnet das „Hauptprogramm“, das z.B. die `onCreate`-Methoden der Activity-Klassen aufruft bzw. die Benutzerinteraktion (Auslesen von Sensordaten, Klicken auf Bildschirm, etc.)

verwaltet. Mir müssen also für die Netzwerkverbindung einen eigenen Thread anlegen - wir nennen diesen im Folgenden `BefehlSenderThread`.

Beim Anlegen des `Socket` auf Seiten des Roboters (Client) müssen wir weiter nichts beachten, wir können hier vorgehen wie im vorangegangenen Kapitel.

10.2.2 Interprozesskommunikation

Die LegoPilot-App besteht also aus zwei Threads. Einmal dem `UIThread`, der für die GUI und Interaktion mit dem Benutzer zuständig ist und dem `BefehlSenderThread`, der die Netzwerkverbindung verwaltet. Diese beiden Threads müssen nun geeignet miteinander kommunizieren, d.h. falls der Benutzer auf die Pfeiltaste vorwärts klickt, soll dies der `UIThread` dem `BefehlSenderThread` mitteilen, damit dieser wiederum den entsprechenden Befehl über die Netzwerkverbindung an den Roboter schicken kann. Wir sprechen in diesem Fall von Interprozesskommunikation.
Wirft man einen genaueren Blick auf unser Szenario, bietet sich hier das **Erzeuger-Verbraucher-Prinzip** an, wobei in unserem Fall der `UIThread` der Erzeuger und der `BefehlSenderThread` der Verbraucher ist. Dabei blockiert der `BefehlSenderThread` solange, bis er vom `UIThread` einen neuen Befehl übermittelt bekommt. Gleichzeitig umgehen wir mit dieser Art der Interprozesskommunikation auch das Problem der Nebenläufigkeit. Hierfür gibt es bereits eine vordefinierte Datenstruktur - die `LinkedBlockingQueue` der Schnittstelle `BlockingQueue`. Die Methode `take()` blockiert automatisch solange, bis in der Warteschlange ein Wert enthalten ist. Ist dies der Fall, liest die Methode diesen Wert aus und löscht diesen aus der Warteschlange.
In Abbildung 10.2 sind unsere bisherigen Überlegungen noch einmal visualisiert:

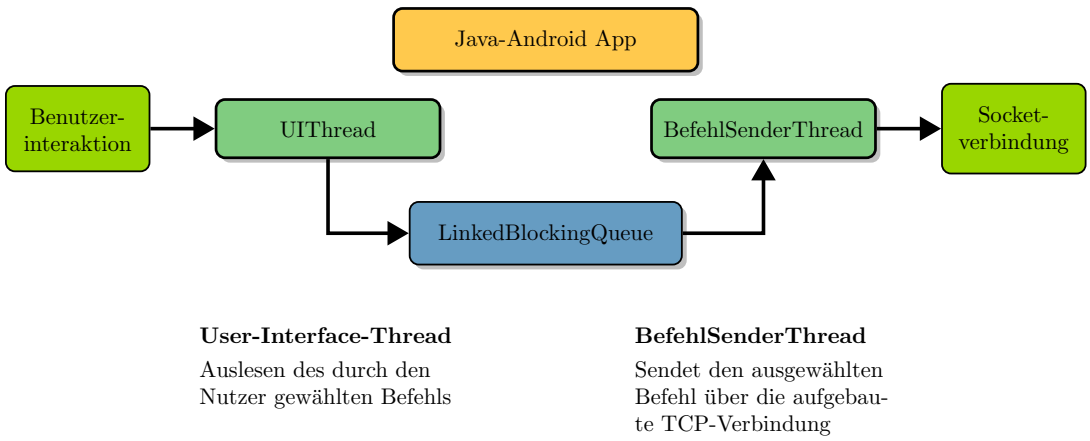


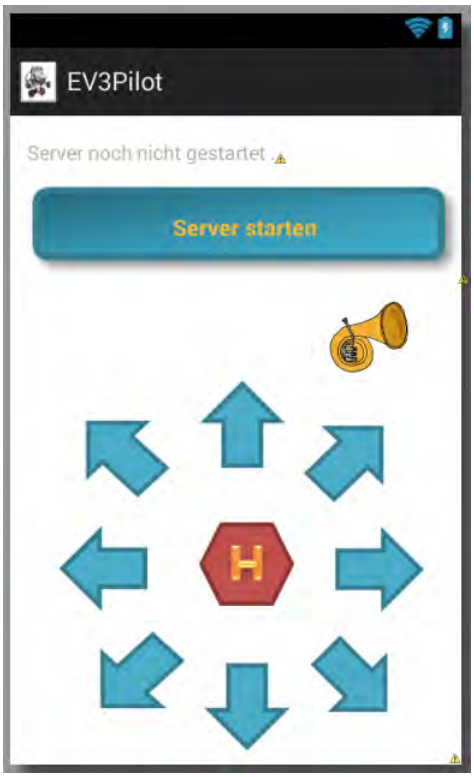
Abbildung 10.2: Interprozesskommunikation nach dem Erzeuger-Verbraucher-Prinzip über eine `BlockingQueue`.

10.3 Erstellen der LegoPilot-App

Wir erstellen zuerst den Applikation für das Smartphone. Dazu legen wir in Eclipse ein Java-Applikation-Projekt mit dem Namen `LegoPilot` an und fügen in der `AndroidManifest.xml`-Datei die in Abschnitt 10.2.1 erwähnte Zeile hinzu, so dass unsere Applikation später eine Verbindung über ein Netzwerk mit dem Roboter herstellen kann.

10.3.1 Layout der App

Zunächst kümmern wir um das Layout der LegoPilot App. Wir brauchen dazu jede Menge Schaltflächen und ein Objekt der Klasse `EditText`, um anzeigen zu können, ob der Server auf dem Smartphone gestartet wurde oder nicht. Ihre Layout-Datei sollte ungefähr wie folgt aussehen:



10.3.2 Enum-Klasse Befehle

Um die Befehle, die durch Benutzereingabe entstehen besser verwalten zu können, erstellen wir uns eine `Enum`-Klasse `Befehle`, in der wir die Werte `FORWARD` für 'Vorwärts fahren', `REVERSE` für 'Rückwärts fahren', `RIGHT` für 'Nach rechts drehen', `LEFT` für 'Nach links drehen', `FORWARD_RIGHT` für 'Nach rechts vorne fahren', `FORWARD_LEFT` für 'Nach links vorne fahren', `REVERSE_RIGHT` für 'Nach rechts hinten fahren', `REVERSE_LEFT` für 'Nach links hinten fahren', `HALT` für 'Stehen bleiben', `HORN` für 'Ton ausgeben' und `EXIT` für 'Verbindung trennen' festlegen:

Code 10.2: Enum-Klasse Befehl.java

```
1 package com.example.ev3pilot;
2
3 public enum Befehl {
4     FORWARD, REVERSE, LEFT, RIGHT, FORWARD_LEFT, FORWARD_RIGHT, REVERSE_RIGHT, REVERSE_LEFT, HALT, HORN,
          EXIT
5 }
```

10.3.3 Die Klasse MainActivity

In der MainActivity-Klasse benötigen wir wie in Abschnitt 10.2.2 als Attribut ein Objekt der Klasse `LinkedBlockingQueue`. Zudem erstellen wir noch Attribute für je ein Objekt des `Enum` Befehl, der Klasse Button und der Klasse `TextView`, sowie einen boolean `connected`. Die Attribute werden in der `onCreate`-Methode initialisiert:

Code 10.3: Enum-Klasse Befehl.java

```
public class MainActivity extends Activity {

    /*
     * BlockingQueue fuer die Interprozesskommunikation zwischen UIThread und
     * "Connection-Thread"
     */
    private LinkedBlockingQueue<Befehl> queue;
    private Befehl aktuellerBefehl;
    private boolean connected = false;
    private Button b;
    private TextView tv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv = (TextView) findViewById(R.id.textView1);
        b = (Button) findViewById(R.id.serverbutton);

        /*
         * Initialisierung der BlockingQueue - ein Platz fuer den aktuellen
         * Befehl
         */
        queue = new LinkedBlockingQueue<Befehl>(1);
        aktuellerBefehl = Befehl.HALT;
    }
}
```

Das Attribut `b` verweist auf die Schaltfläche, mit der man den Server starten kann. Unsere `LinkedBlockingQueue` soll einen Eintrag des `Enum` Befehl haben. Das Attribut `connected` gibt Rückschlüsse darauf, ob eine Verbindung zum Roboter aufgebaut wurde oder nicht - folglich wird es mit `false` initialisiert.

Nun benötigen wir Methoden, die aufgerufen werden, wenn auf die Schaltflächen mit den Pfeilen bzw. Stopp-Symbol bzw. Hupe geklickt wird. Wir erstellen für jede Schaltfläche eine eigene Methode und weisen diese dem entsprechenden Button-Objekt in der Layout-Datei über den Befehl `android:onClick="..."` zu.
In diesen Methoden überprüfen wir, ob überhaupt eine Verbindung zum Roboter hergestellt ist und ob sich der vom Benutzer ausgewählte Befehl vom aktuellen Befehl unterscheidet. Ist dies der Fall, wird der neue Befehl in die `LinkedBlockingQueue` eingefügt (vgl. Abbildung 10.2).
Im Folgenden exemplarisch zwei Methoden für die Vorwärts- und Anhalten-Schaltfläche:

Code 10.4: Methoden, die beim Klicken auf die Vorwärts-Schaltfläche bzw. Anhalten-Schaltfläche aufgerufen werden.

```
/**
* Wird aufgerufen, wenn auf die FWD-Pfeiltaste geklickt wird Falls bislang
* ein anderer Befehl ausgefuehrt wird, wird der FWD-Befehl in die Queue
* eingefuegt
*
* @param view
*/
public void ButtonFWDPressed(View view) {
    if (connected && aktuellerBefehl != Befehl.FORWARD) {
        try {
            queue.put(Befehl.FORWARD);
            aktuellerBefehl = Befehl.FORWARD;
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
```

```
            e.printStackTrace();
        }
    }
}

/**
 * Wird aufgerufen, wenn auf die HALT-Taste geklickt wird Falls bislang ein
 * anderer Befehl ausgefuehrt wird, wird der HALT-Befehl in die Queue
 * eingefuegt
 *
 * @param view
*/
public void ButtonHALTPressed(View view) {
    if (connected && aktuellerBefehl != Befehl.HALT) {
        try {
            queue.put(Befehl.HALT);
            aktuellerBefehl = Befehl.HALT;
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Wir benötigen nun noch eine Methode, die aufgerufen wird, falls auf die 'Server starten'-Schaltfläche geklickt wird. Falls der Server noch nicht gestartet wurde, müssen wir einen neuen `ServerSocket` initialisieren. Wie bereits in Abschnitt 10.2.1 erwähnt, können wir dies nicht auf dem `UIThread` machen, sondern müssen einen neuen Thread für die Netzwerkverbindung anlegen. Darum kümmern wir uns im nächsten Unterabschnitt. Wird auf die Schaltfläche geklickt, obwohl der Server bereits gestartet war, wird der Server geschlossen. In diesem Fall fügen wir noch den Befehl `EXIT` in unsere `LinkedBlockingQueue` ein:

Code 10.5: Methoden, die beim Klicken auf die Vorwärts-Schaltfläche bzw. Anhalten-Schaltfläche aufgerufen werden.

```
/**
* Wird aufgerufen wenn auf die Verbindung herstellen / Verbindung trennen
* Schaltflaeche geklickt wird. Wurde der Server noch nicht gestartet, wird
* ein neuer "Connection-Thread" angelegt (Socket-Verbindung kann nicht auf
* dem UI-Thread angelegt werden). Dieser Thead ist fuer die
* Socket-Connection und das Schicken der Befehle verantwortlich. Wurde der
* Server gestartet, kann man durch erneutes Druecken auf die Schaltflaeche
* die Verbindung beenden
*
* @param view
*/
public void ServerButtonPressed(View view) {
    if (!connected) {
        tv.setText("Server wird gestartet ...");
        tv.setText("Server bereit. Warten auf Client ...");
        // TODO: neuen Thread anlegen und starten
    } else {
        try {
            queue.put(Befehl.EXIT);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        b.setText("Server starten");
        tv.setText("Server noch nicht gestartet");
        connected = false;
    }
```

```
28 }
```

10.3.4 Die Klasse BefehlSenderThread

In diesem Unterabschnitt erstellen wir die Klasse `BefehlSenderThread`, die von der Klasse `Thread` erbt:

Code 10.6: Die Klasse BefehlSenderThread

```
1  package com.example.ev3pilot;
2
3  public class BefehlSenderThread extends Thread {
4
5      public void run() {
6
7      }
8  }
```

In diesem Thread verwalten wir die Netzwerkverbindung zum Roboter. Wir müssen also zunächst ein Objekt der Klasse `ServerSocket` erstellen, das wartet, bis sich ein Client (in unserem Fall der EV3-Roboter) mit diesem verbindet. Wurde eine Verbindung hergestellt, werden je nach Befehl in der `LinkedBlockingQueue` (der BefehlSenderThread wartet immer so lange, bis ein neuer Befehl in die Queue eingefügt wird) entsprechende Nachrichten über die Socket-Verbindung an den Roboter gesendet.
Somit benötigt diese Klasse auf jeden Fall Zugriff auf die `LinkedBlockingQueue` der Activity-Klasse. Folglich muss diese im Konstruktor übergeben werden. Weiter müssen wir das Attribut `connected` der Activity-Klasse auf `true` setzen, falls sich der Roboter mit der App verbindet. Wurde eine Verbindung hergestellt, müssen wir auch die Beschriftung unseres `EditText`-Objekts bzw. der Verbinden-Schaltfläche ändern - folglich werden auch diese im Konstruktor übergeben:

Code 10.7: Attribute und Konstruktuor der Klasse `BefehlSenderThread`

```
1  /*
2  * LinkedBlockingQueue fuer die Interprozesskommunikation
3  */
4  private LinkedBlockingQueue<Befehl> queue;
5  /*
6   * Objekte fuer die zugehoerige Activity-Klasse, sowie zwei deren
7   * View-Objekte
8   */
9  private MainActivity activity;
10 private TextView tv;
11 private Button b;
12
13 /**
14 * Konstruktor - legt einen neuen BefehlSenderThread an. Bekommt Referenz
15 * auf
16 *
17 * @param activity die zugehoerige Activity-Klasse,
18 * @param tv TextView-Objekt der Activity-Klasse,
19 * @param b Button-Objekt der Activity-Klasse,
20 * @param queue BlockingQueue
21 */
22 public BefehlSenderThread(MainActivity activity, TextView tv, Button b, LinkedBlockingQueue<Befehl> queue) {
23     this.activity = activity;
24     this.queue = queue;
25     this.killed = false;
26     this.tv = tv;
27     this.b = b;
28 }
```

In der `run`-Methode erstellen wir nun zunächst einen ServerSocket auf Port 2610 (hier hätte man auch jeden beliebigen anderen freien Port verwenden können). Anschließend warten wir darauf, bis sich der Roboter mit unserem Programm verbindet. Dies geschieht mittels des Befehls `serverSocket.accept()`, der solange blockiert, bis sich ein Socket mit dem ServerSocket verbunden hat. Wurde eine Verbindung hergestellt, müssen wir das Attribut `connected` auf `true` setzen. Dazu erstellen wir eine Methode `setConnected()` in der `MainActivity`-Klasse, die `connected` auf `true` setzt und rufen diese in unserem Thread auf:

Code 10.8: Die `run`-Methode der Klasse `BefehlSenderThread`

```
1  private Socket client;
2
3  /**
4   * Run-Methode des BefehlSenderThreads
5   */
6  public void run() {
7      try {
8          // Initialisieren des ServerSockets
9          serverSocket = new ServerSocket(2610);
10         // Methode accept() blockiert solange, bis die Verbindung zu einem Client aufgebaut wurde
11         client = serverSocket.accept();
12         activity.setConnected();
13
14         // TODO: Beschriftung der Schaltflaeche und des ExitText-Objektes aendern
15
16     } catch (IOException e1) {
17         // TODO Auto-generated catch block
18         e1.printStackTrace();
19     }
20 }
```

An der mit `TODO` gekennzeichneten Stelle möchten wir noch die Beschriftung der Verbinden-Schaltfläche und des EditText-Objekts ändern. Hier die intuitive Lösung ist `b.setText("...")` und `tv.setText("...")`. Hier macht uns Java-Android allerdings einen Strich durch die Rechnung. Man kann auf View-Objekte nur innerhalb des `UIThreads` zugreifen - wir befinden uns gerade allerdings in einem anderen Thread und erhalten folglich eine Fehlermeldung.
Um diese Problematik zu umgehen, gibt es für jedes Objekt einer `Activity`-Klasse die Methode `runOnUiThread`, die folgende Syntax hat:

Code 10.9: Zugriff auf View-Objekte aus einem anderen Thread über die Methode `runOnUiThread`

```
1  runOnUiThread(new Runnable() {
2      public void run() {
3          // Befehle, die vom UIThread durchgefuehrt werden sollen
4      }
5  });
```

Wir können diese Methode wie folgt nutzen, um die Beschriftung der Schaltfläche und des `EditText`-Objekts aus dem `BefehlSenderThread` heraus zu ändern:

Code 10.10: Verwendung der Methode `runOnUiThread` um die Beschriftung der Schaltfläche und des `EditText`-Objekts zu ändern.

```
1  // Nur der UIThread hat Zugriff auf die View-Objekte der Activity-Klasse. Hierfuer gibt es die Methode
       runOnUiThread
2  activity.runOnUiThread(new Runnable() {
3      public void run() {
4          tv.setText("Verbindung hergestellt...");
```

```
5           b.setText("Verbindung trennen");
6       }
7   });
```

Diese Befehle müssen wir nun noch an geeigneter Stelle in unsere `run`-Methode einfügen.

Zusammenfassung:

In der `run`-Methode haben wir unser ServerSocket-Objekt initialisiert und warten darauf, bis sich ein Client anmeldet. Ist dies der Fall, setzen wir das Attribut `connected` der `MainActivity`-Klasse auf `true` und ändern die Beschriftungen der Verbinden-Schaltfläche und des `EditText`-Objektes.
Nun müssen wir uns noch um das Verschicken der Nachrichten an den Roboter kümmern. Die Nachrichten erhalten wir vom `UIThread` über die `LinkedBlockingQueue`. Dieser fügt, immer wenn sich der aktuelle Befehl durch Benutzereingabe verändert, den neuen Befehl in die `LinkedBlockingQueue` ein. Wir müssen in der `run`-Methode des `BefehlSenderThreads` somit immer darauf warten, bis ein neuer Befehl in die Queue eingefügt wird und diesen dann über die Socket-Verbindung schicken. Hierbei unterstützt und die Datenstruktur `LinkedBlockingQueue`, da die Methode `take()` automatisch blockiert, solange kein Wert in der Queue abgelegt wurde. Wir möchten so lange Befehle schicken, wie die Verbindung aufgebaut ist. Dazu fügen wir noch ein `boolean`-Attribut `killed` ein, das wir auf `true` setzen, falls die Verbindung unterbrochen wird.
Für das Senden der Befehle über die Socket-Connection erstellen wir eine Hilfsmethode `sendeBefehl`, um die wir uns im Anschluss kümmern.
Um Befehle schicken zu können, benötigen wir wieder ein Objekt der Klasse `PrintWriter`, das wir, wie aus vorangegangenen Kapitel bekannt, geeignet initialisieren:

Code 10.11: Falls vom UIThread ein neuer Befehl in die Queue eingefuegt wird, wird dieser sofort wieder herausgenommen und anschließend ueber die Socket-Verbindung geschickt.

```
1   private PrintWriter pWriter;
2
3   /**
4   * Run-Methode des BefehlSenderThreads
5   */
6   public void run() {
7       try {
8           // Initialisieren des ServerSockets
9           serverSocket = new ServerSocket(2610);
10
11          ...
12
13          // Initialisieren des PrintWriters
14          pWriter = new PrintWriter(new OutputStreamWriter(
15          client.getOutputStream()));
16      } catch (IOException e1) {
17          // TODO Auto-generated catch block
18          e1.printStackTrace();
19      }
20      // Solange die Verbindung zum Client besteht werden nun Befehle gesendet
21      while (!killed) {
22          try {
23              // Methode take der Klasse LinkedBlockingQueue ist blockierend. Hier wird solange gewartet, bis
                    vom UIThread ein neuer Befehl in die LinkedBlockinQueue eingefuegt wurde (Producer-Consumer
                    -Prinzip)
24              sendeBefehl(queue.take());
25          } catch (InterruptedException | IOException e) {
26              // TODO Auto-generated catch block
27              e.printStackTrace();
```

```
28          }
29      }
30  }
31
32  private void sendeBefehl(Befehl befehl) {
33      // TODO
34  }
```

Zum besseren Verständnis ist die Interaktion der beiden Threads für den Fall, dass der Benutzer zunächst die Vorwärts-Taste und anschließend die Halt-Taste in folgender Grafik visualisiert:

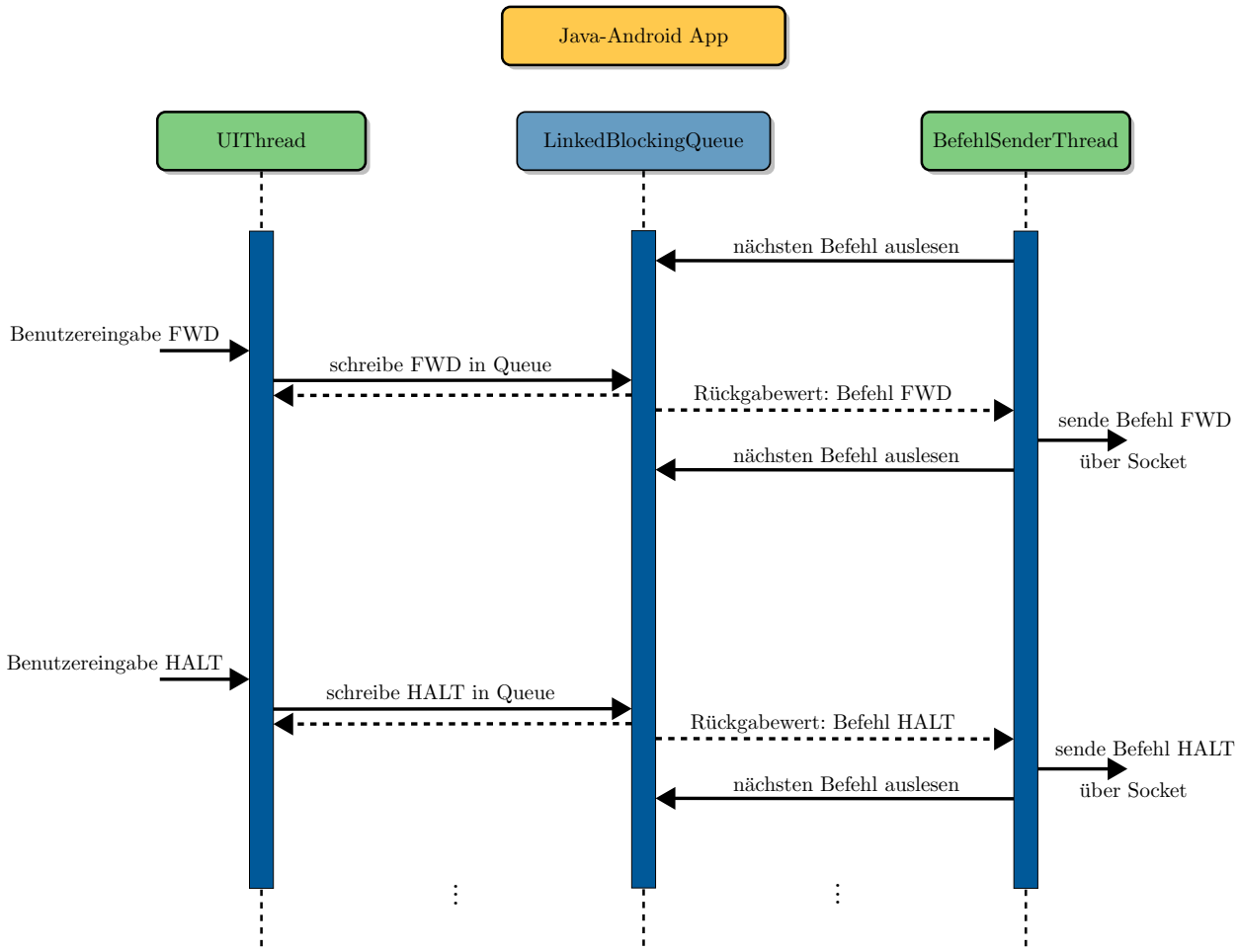


Abbildung 10.3: Interaktion der beiden Threads für die Benutzereingaben Vorwärts und Halt.

Die Hilfsmethode `sendeBefehl(Befehl befehl)`

Da wir über die Socket-Verbindung nicht ohne weiteres unsere Enums verschicken können, wandeln wir diese in Strings um. Dazu verwenden wir eine `switch / case`-Anweisung. Den String schicken wir anschließend mit Hilfe des PrintWriters über die Socket-Verbindung:

Code 10.12: Die Hilfsmethode `sendeBefehl(Befehl befehl)`

```
1  /**
2  * Methode, die das Enum Befehl in die zugehoerige String-Repraesentation
3  * umwandelt und ueber die Socket-Connection verschickt
```

```
4   *
5   * @param befehl umzuwandelnder Befehl
6   * @throws IOException
7   */
8   private void sendeBefehl(Befehl befehl) throws IOException {
9
10      String nachricht = "HALT";
11      switch (befehl) {
12      case FORWARD:
13          nachricht = "FWD";
14          break;
15      case FORWARD_LEFT:
16          nachricht = "FWD_LEFT";
17          break;
18      case FORWARD_RIGHT:
19          nachricht = "FWD_RIGHT";
20          break;
21      case HALT:
22          nachricht = "HALT";
23          break;
24      case HORN:
25          nachricht = "HORN";
26          break;
27      case LEFT:
28          nachricht = "LEFT";
29          break;
30      case REVERSE:
31          nachricht = "REVERSE";
32          break;
33      case RIGHT:
34          nachricht = "RIGHT";
35          break;
36      case REVERSE_RIGHT:
37          nachricht = "REV_RIGHT";
38          break;
39      case REVERSE_LEFT:
40          nachricht = "REV_LEFT";
41          break;
42      case EXIT:
43          nachricht = "EXIT";
44          killed = true;
45      default:
46          break;
47      }
48
49      // String-Repraesentation des Befehls wird ueber die Socket-Connection
50      // gesendet
51      pWriter.write(nachricht + "\n");
52      pWriter.flush();
53  }
```

Kapitel 11

Lösungen

Überblick:

11.1 Kapitel 4 - Einführungsaufgabe

11.1.1 Lösung zu Übung 1

(a) Sie müssen hier im Vergleich zur Einführungsaufgabe nur die Methode `changeBackgroundColor` ändern und etwa eine `boolsche` Variable einführen, mit der Sie die Fallunterscheidung zwischen weiß und gelb realisieren können:

```
boolean istWeiss = true;

/**
* Methode zum Aendern der Hintergrundfarbe
* @param view
*/
public void changeBackgroundColor(View view) {
    RelativeLayout rl = (RelativeLayout) findViewById(R.id.container);

    if (istWeiss) {
        rl.setBackgroundColor(Color.YELLOW);
        istWeiss = false;
    } else {
        rl.setBackgroundColor(Color.WHITE);
        istWeiss = true;
    }
}
```

(b) Dazu benötigen wir in der Activity-Klasse Zugriff auf das Button-Objekt. Wir ändern dann in der `if/else`-Kontrollstruktur die Beschriftung der Schaltfläche:

```
boolean istWeiss = true;

/**
* Methode zum Aendern der Hintergrundfarbe und der Button-Beschriftung
* @param view
*/
public void changeBackgroundColor(View view) {
  RelativeLayout rl = (RelativeLayout) findViewById(R.id.container);
  Button b = (Button) findViewById(R.id.changeColorButton);

  if (istWeiss) {
```

```
12      rl.setBackgroundColor(Color.YELLOW);
13      b.setText("Hintergrundfarbe auf weiss setzen");
14      istWeiss = false;
15    } else {
16      rl.setBackgroundColor(Color.WHITE);
17      b.setText("Hintergrundfarbe auf gelb setzen");
18      istWeiss = true;
19    }
20  }
```

(c) Hier gibt es ganz verschiedene Möglichkeiten. Man könnte sich etwa sieben verschiedene Farben in einem Array speichern und immer zufällig eine Farbe zur Hintergrundfarbe machen:

```
1  Random r = new Random();
2  int[] farben = {Color.BLUE, Color.CYAN, Color.GREEN, Color.LTGRAY, Color.MAGENTA, Color.RED, Color.
       WHITE};
3
4  /**
5  * Methode zum Aendern der Hintergrundarbe auf eine zufaellige Farbe einer bestimmten Vorauwahl
6  * @param view
7  */
8  public void changeBackgroundColor(View view) {
9      RelativeLayout rl = (RelativeLayout) findViewById(R.id.container);
10     rl.setBackgroundColor(farben[r.nextInt(7)]);
11 }
12 \end{enumerate}
```

Man könnte die Farben aber auch wirklich total willkürlich generieren lassen:

```
1  /**
2  * Methode zum Aendern der Hintergrundarbe auf eine zufaellige Farbe einer bestimmten Vorauwahl
3  * @param view
4  */
5  public void changeBackgroundColor(View view) {
6     RelativeLayout rl = (RelativeLayout) findViewById(R.id.container);
7     rl.setBackgroundColor(Color.rgb(r.nextInt(256), r.nextInt(256), r.nextInt(256)));
8  }
```

11.1.2 Lösung zu Übung 2

Zunächst erstellen wir in der Layout-Datei `activity_main.xml` ein Objekt der Klasse `EditText` und geben diesem die `ID` *edittext*. Falls Sie hiermit Probleme haben, lesen Sie noch einmal Abschnitt 4.2.3 durch.

In der Methode `changeBackgroundColor` überprüfen wir nun zunächst, ob in das Textfeld was eingegeben wurde und nehmen dann die gewünschten Änderungen vor:

```
1  /**
2   * Methode zum Aendern der Hintergrundarbe und der Beschriftung der Schaltflaeche
3   * @param view
4   */
5  public void changeBackgroundColor(View view) {
6      RelativeLayout rl = (RelativeLayout) findViewById(R.id.container);
7      EditText et = (EditText) findViewById(R.id.edittext);
8      Button b = (Button) findViewById(R.id.changeColorButton);
9      String s = et.getText().toString();
```

```
10
11      // Wurde in das Textfeld ein Text eingegeben ...
12      if (!s.equals("")) {
13          // ... wird dieser als neue Beschriftung unserer Schaltflaeche festgelegt und die Hintergrundfarbe
                geaendert
14          b.setText(s);
15          rl.setBackgroundColor(Color.rgb(r.nextInt(256), r.nextInt(256), r.nextInt(256)));
16      }
17  }
```

11.2 Kapitel 5 - Englisch-Vokabel-App

11.2.1 Lösung zu Aufgabe 1

(a) Die beiden neuen Activity-Klassen erstellen Sie wie in Abschnitt 5.5 beschrieben. Diese sollten wie folgt aussehen:

Code 11.1: Activity-Klasse `AddActivity.java`

```
1  package com.example.englisch_vokabeln;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class AddActivity extends Activity {
7
8    @Override
9    protected void onCreate(Bundle savedInstanceState) {
10   super.onCreate(savedInstanceState);
11   setContentView(R.layout.activity_add);
12   }
13 }
```

Code 11.2: Activity-Klasse `VocTestActivity.java`

```
1  package com.example.englisch_vokabeln;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class VocTestActivity extends Activity {
7
8    @Override
9    protected void onCreate(Bundle savedInstanceState) {
10     super.onCreate(savedInstanceState);
11     setContentView(R.layout.activity_voctest);
12   }
13 }
```

(b) Nun müssen wir in der Klasse `SelectionActivity` noch festlegen, dass die entsprechende Activity bei Klicken auf die zugehörige Schaltfläche aufgerufen wird.
Dazu erstellen wir in der `SelectionActivity`-Klasse folgende zwei Methoden:

Code 11.3: Neue Methoden in der Klasse `SelectionActivity.java`

```
1  public void startAddActivity(View view) {
2    Intent myIntent = new Intent(this, AddActivity.class);
3    startActivity(myIntent);
4  }
5
6  public void startVocTestActivity(View view) {
7    Intent myIntent = new Intent(this, VocTestActivity.class);
8    startActivity(myIntent);
9  }
```

Diese Methoden weisen wir nun in der XML-Ansicht der `activity_selection`-Datei den entsprechenden Schaltflächen zu. Falls Sie nicht mehr wissen wie das geht, können Sie in Abschnitt 4.4 von Kapitel 4 nachschauen.

11.2.2 Lösung zu Aufgabe 2

Die Tabelle kann mit folgendem SQLite-Befehl erzeugt werden:

```
CREATE TABLE IF NOT EXISTS vocTable(
    deutsch  TEXT NOT NULL,
    englisch TEXT NOT NULL);
```

11.2.3 Lösung zu Aufgabe 3

(a) Das neue Vokabelpaar kann mit folgendem SQL-Befehl in die Tabelle eingefügt werden:

```
INSERT INTO vocTable(deutsch,englisch)
    VALUES ('deutscheBedeutung','englischeBedeutung');
```

(b) Die Methode `addEntry` zum Einfügen eines neuen Vokabelpaares in die Tabelle `vocTable`:

```
1  /**
2   * Methode addEntry zum Einfuegen eines neuen Vokabelpaares in die Tabelle vocTable
3   */
4  public void addEntry(String deutscheBed, String englischeBed) {
5    db.execsql("insert into vocTable(deutsch,englisch) values ('" + deutscheBed + "','" + englischeBed +
         "');");
6  }
```

11.2.4 Lösung zu Aufgabe 4

(a) Die Methode `addButtonClicked`:

```
1  public void addButtonClicked(View view) {
2
3  }
```

Beachten Sie den Aufrufparameter `View` und vergessen Sie nicht, die Methode in der XML-Datei dem Button zuzuweisen.

(b) Zunächst stellen wir via `findViewById` „eine Verbindung“ zu den beiden EditText-Objekten aus der Layout-Datei her und lesen den Text aus. Anschließend überprüfen wir, ob bei beiden Bedeutungen was eingetragen ist oder nicht. Ist dies nicht der Fall, geben wir einen Toast als Benachrichtigung aus:

```
public void addButtonClicked(View view) {
    // Auslesen des eingegebenen Textes aus den Eingabefeldern
    String deutscheBedeutung = ((EditText) findViewById(R.id.editGerman)).getText().toString();
    String englischeBedeutung = ((EditText) findViewById(R.id.editEnglish)).getText().toString();

    // War eines oder beide Eingabefelder leer, wird eine Fehlermeldung angezeigt
    if(deutscheBedeutung.equals("") || englischeBedeutung.equals("")) {
        Toast.makeText(this, "Bitte alle Felder ausfuellen!", Toast.LENGTH_LONG).show();
    } else {

    }
}
```

(c) Speichern der Vokabel:

```
public void addButtonClicked(View view) {
    // Auslesen des eingegebenen Textes aus den Eingabefeldern
    String deutscheBedeutung = ((EditText) findViewById(R.id.editGerman)).getText().toString();
    String englischeBedeutung = ((EditText) findViewById(R.id.editEnglish)).getText().toString();

    // War eines oder beide Eingabefelder leer, wird eine Fehlermeldung angezeigt
    if(deutscheBedeutung.equals("") || englischeBedeutung.equals("")) {
        Toast.makeText(this, "Bitte alle Felder ausfuellen!", Toast.LENGTH_LONG).show();
    } else {
        //Schreibrechte beantragen, Vokabel speichern, Berechtigungen entziehen
        dbHandler.openWrite()
        dbHandler.addEntry(deutscheBedeutung,englischeBedeutung);
        dbHandler.close();

        Toast.makeText(this, "Vokabel gespeichert", Toast.LENGTH_LONG).show();
    }
}
```

11.2.5 Lösung zu Aufgabe 5

Die Anzahl der Einträge kann mit folgendem `sql`-Befehl ermittelt werden:

```
SELECT COUNT(deutsch)
FROM vocTable
```

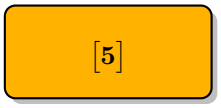
11.2.6 Lösung zu Aufgabe 6

(a) Da wir die Anzahl der Einträge der Tabelle `vocTable` zurückgeben möchten, wählen wir als Rückgabetyp `int`.

(b) In Aufgabe 5 haben wir bereits ein geeignetes SQL-Statement erstellt. Dies führen wir mit dem Befehl `db.rawQuery ( .. )` aus und speichern das Ergebnis in einem Cursor-Objekt:

```
Cursor c = db.rawQuery("select count(*) from vocTable;");
```

(c) Die SQL-Anfrage liefert genau einen Wert, d.h. eine Zeile mit einer Spalte Folglich hat unser Cursor-Objekt ein Element mit einem Eintrag. Hat die Tabelle `vocTable` z.B. fünf Elemente, hat das Cursor-Objekt folgenden Aufbau:



Um also die gesuchte Anzahl aus dem Cursor-Objekt auslesen zu können, müssen wir zunächst den Zeiger auf das erste (und einzige) Element des Cursors setzen und dann auf den Eintrag an Position 0 zugreifen:

```
if (c.moveToFirst()) {
    String anzahl = c.getString(0);
}
```

Folglich hat die Methode `tableSize` folgenden Aufbau:

```
public int tableSize() {
    Cursor c = db.rawQuery("select count(*) from vocTable;");
    String size;
    //Zeiger auf erstes Element des Cursors setzen und Eintrag an Position 0 auslesen
    if (c.moveToFirst()) {
        size = c.getString(0);
    }
    //String muss noch in einen Integer umgewandelt werden
    return Integer.parseInt(size);
}
```

11.2.7 Lösung zu Aufgabe 7

(a) Da die Beschriftung immer bei Aufrufen der Activity-Klasse geändert werden soll, machen wir dies in der `onCreate`-Methode. Die Variable `showVoc` setzen wir auf `true`:

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

    showVoc = true;
    // Entsprechende Beschriftung der Anzeigen Schaltflaeche bei Aufrufen der Activity-Klasse
    ((Button) findViewById(R.id.vocButton)).setText("Abfrage starten");
    // Auf der Karteikarte soll nichts angezeigt werden
    ((Button) findViewById(R.id.vocKarte)).setText("");
}
```

(b) Hier bietet es sich an, die Beschriftung in der `buttonClicked`-Methode zu ändern, nachdem die private Hilfsmethode `loesungAnzeigen` aufgerufen wurde:

```
/**
* Wird bei Klicken auf die Anzeigen-Schaltflaeche aufgerufen
* @param view
*/
public void buttonClicked(View view) {
    if(newVoc) {
        neueVokabelAnzeigen();
    } else {
        loesungAnzeigen();
```

```
10          // Entsprechende Beschriftung der Anzeigen-Schaltflaeche, falls eine Loesung angezeigt wird
11          ((Button) findViewById(R.id.vocButton)).setText("Neue Vokabel anzeigen");
12      }
13  }
```

11.2.8 Lösung zu Aufgabe 8

(a) Da wir die deutsche und englische Bedeutung einer Vokabel zurückgeben wollen, wählen wir als Rückgabetyp ein String-Array.

(b) Das Cursor-Objekt hat ein Element mit zwei Einträgen:

$$[\textbf{deutscheBed}, \textbf{englischeBed}]$$

(c) In der Methode `getRandomData()` holen wir uns zunächst die deutsche und englische Bedeutung einer zufällig ausgewählten Vokabel aus der Datenbank. Die beiden Bedeutungen werden in ein Cursor-Objekt gespeichert, das von obiger Form ist. Um die beiden Bedeutungen auslesen zu können, setzen wir den Zeiger auf das erste Element und greifen dann auf die Einträge an der nullten und ersten Position zu. Diese schreiben wir in ein String-Array, welches wir zurückgeben:

```
1   /**
2    * Die Methode holt einen zufaelligen Datensatz aus der Tabelle vocTable der Datenbank vocDB.
3    * Ist die Datenbank nicht leer, hat der Cursor genau ein Wertepaar (deutsche Bedeutung,
4    * englische Bedeutung). Diese beiden String-Objekte werden in einem String-Array zurueckgegeben
5    */
6   public String[] getRandomData() {
7       String[] vokabel = new String[2];
8       Cursor c = db.rawQuery("select deutsch,englisch from vocTable order by random() limit 1;", null);
9       // Falls der Cursor Eintraege hat, wird der Zeiger auf das erste Element gesetzt
10      if (c.moveToNext()) {
11          // Zugriff auf die deutsche und englische Bedeutung und Abspeichern im String-Array
12          vokabel[0] = c.getString(0);
13          vokabel[1] = c.getString(1);
14      } else {
15          // Ist keine Vokabel in der Datenbank wird null zurueckgegeben
16          return null;
17      }
18      return vokabel;
19  }
```

11.2.9 Lösung zu Aufgabe 9

Zunächst brauchen wir Zugriff auf die beiden Switch-Objekte, um überprüfen zu können, ob diese auf `ON` oder `OFF` stehen:

```
1   Switch deutschenglisch = (Switch) findViewById(R.id.deutschenglischswitch);
2   Switch englischdeutsch = (Switch) findViewById(R.id.englischdeutschswitch);
```

Als nächstes überprüfen wir, ob mindestens eines der beiden Switch-Objekte auf `ON` steht. Dies funktioniert am einfachsten mit dem logischen-Operator `OR`:

```
if (deutschenglisch.isChecked() || englischdeutsch.isChecked()) {

} else {

}
```

Nun kümmern wir uns noch darum, dass eine Fehlermeldung ausgegeben wird, falls keine Abfragerichtung ausgewählt ist und ein zufälliges Vokabelpaar aus der Datenbank geholt wird, falls schon. Hierzu verwenden wir die Methode `getRandomData()` der `DataHandler` Klasse:

```
private void neueVokabelAnzeigen() {
    Switch deutschenglisch = (Switch) findViewById(R.id.deutschenglischswitch);
    Switch englischdeutsch = (Switch) findViewById(R.id.englischdeutschswitch);

    if (deutschenglisch.isChecked() || englischdeutsch.isChecked()) {
        // Zufaelliges Vokabelpaar aus der Datenbank holen
        dbHandler.openRead();
        String[] vokabel = dbHandler.getRandomData();
        dbHandler.close();

    } else {
        Toast.makeText(this, "Bitte mindestens eine Abfragerichtung auswaehlen", Toast.LENGTH_LONG).show();
    }

}
```

11.2.10 Lösung zu Aufgabe 10

Wir wissen, dass im `if`-Zweig mindestens eine Abfragerichtung ausgewählt ist. Nun testen wir zunächst, ob beide ausgewählt sind und falls dies nicht der Fall ist, welche Abfragerichtung ausgewählt ist:

```
if (deutschenglisch.isChecked() && englischdeutsch.isChecked()) {
    // beide Abfragerichtungen sind ausgewaehlt
} else if (deutschenglisch.isChecked()) {
    // es ist nur die Abfragerichtung Deutsch -> Englisch ausgewaehlt
} else {
    // es ist nur die Abfragerichtung Englisch -> Deutsch ausgewaehlt
}
```

11.2.11 Lösung zu Aufgabe 11

Wir überprüfen zunächst, ob in der Datenbank eine Vokabel gespeichert war. Dies können wir überprüfen, indem wir testen, ob das String-Array `vokabel` `null` ist oder nicht (vgl. Rückgabewerte der Methode `getRandomData`). Wurde ein Vokabelpaar im String-Array `vokabel` gespeichert, zeigen wir je nach Abfragerichtung die eine Bedeutung auf der Karteikarten an, die andere speichern wir in der Variable `loesung`.
Wir setzen noch die Variable `showVoc` auf `false` und ändern wie beschrieben die Beschriftung der *Anzeigen*-Schaltfläche:

```
private String loesung;

```

```
private void neueVokabelAnzeigen() {
    Switch deutschenglisch = (Switch) findViewById(R.id.deutschenglischswitch);
    Switch englischdeutsch = (Switch) findViewById(R.id.englischdeutschswitch);
    Button karteikarte = (Button) findViewById(R.id.karteikarte);
    // ist mindestens eine Abfragerichtung ausgewaehlt?
    if (deutschenglisch.isChecked() || englischdeutsch.isChecked()) {
        // Zufaelliges Vokabelpaar aus der Datenbank holen
        dbHandler.openRead();
        String[] vokabel = dbHandler.getRandomData();
        dbHandler.close();

        if (vokabel != null) {
            // beide Abfragerichtungen sind ausgewaehlt
            if (deutschenglisch.isChecked() && englischdeutsch.isChecked()) {

              // es ist nur die Abfragerichtung Deutsch -> Englisch ausgewaehlt
            } else if (deutschenglisch.isChecked()) {
                // deutsche Bedeutung wird auf Karteikarte angezeigt, englische Bedeutung in loesung
                    gespeichert
                karteikarte.setText(vokabel[0]);
                loesung = vokabel[1];
              // es ist nur die Abfragerichtung Englisch -> Deutsch ausgewaehlt
            } else {
                // englsiche Bedeutung wird auf Karteikarte angezeigt, deutsche Bedeutung in loesung
                    gespeichert
                karteikarte.setText(vokabel[1]);
                loesung = vokabel[0];
            }
            // showVoc wird auf false gesetzt und Beschriftung des Anzeigen-Buttons geaendert
            showVoc = false;
            ((Button) findViewById(R.id.vocButton)).setText("Loesung anzeigen");

        }


    } else {
        Toast.makeText(this, "Bitte mindestens eine Abfragerichtung auswaehlen", Toast.LENGTH_LONG).show();
    }
}
```

11.2.12 Lösung zu Aufgabe 12

Zunächst ergänzen wir im vorgegebenen Code die Befehle, um die eine Bedeutung der Vokabel auf der Karteikarte anzuzeigen und die andere im String `loesung` zu speichern:

```
Random r = new Random();

// Erzeugen eines zufaelligen boolschen Wertes
if (r.nextBoolean()) {
    // bei true wird die deutsche Bedeutung auf der Karteikarte angezeigt
    karteikarte.setText(vokabel[0]);
    loesung = vokabel[1];
} else {
    // bei false wird die englisch Bedeutung auf der Karteikarte angezeigt
    karteikarte.setText(vokabel[1]);
    loesung = vokabel[0];
}
```

Diesen Code-Abschnitt fügen wir nun noch in unsere Methode `neueVokabelAnzeigen` ein:

```
private String loesung;

private void neueVokabelAnzeigen() {
    Switch deutschenglisch = (Switch) findViewById(R.id.deutschenglischswitch);
    Switch englischdeutsch = (Switch) findViewById(R.id.englischdeutschswitch);
    Button karteikarte = (Button) findViewById(R.id.karteikarte);
    // ist mindestens eine Abfragerichtung ausgewaehlt?
    if (deutschenglisch.isChecked() || englischdeutsch.isChecked()) {
        // Zufaelliges Vokabelpaar aus der Datenbank holen
        dbHandler.openRead();
        String[] vokabel = dbHandler.getRandomData();
        dbHandler.close();

        if (vokabel != null) {
            // beide Abfragerichtungen sind ausgewaehlt
            if (deutschenglisch.isChecked() && englischdeutsch.isChecked()) {
                Random r = new Random();

                // Erzeugen eines zufaelligen boolschen Wertes
                if (r.nextBoolean()) {
                    // bei true wird die deutsche Bedeutung auf der Karteikarte angezeigt
                    karteikarte.setText(vokabel[0]);
                    loesung = vokabel[1];
                } else {
                    // bei false wird die englisch Bedeutung auf der Karteikarte angezeigt
                    karteikarte.setText(vokabel[1]);
                    loesung = vokabel[0];
                }
            // es ist nur die Abfragerichtung Deutsch -> Englisch ausgewaehlt
            } else if (deutschenglisch.isChecked()) {
                ...
            } else {
                ...
            }

            // showVoc wird auf false gesetzt und Beschriftung des Anzeigen-Buttons geaendert
            showVoc = false;
            ((Button) findViewById(R.id.vocButton)).setText("Loesung anzeigen");
        }
    } else {
        Toast.makeText(this, "Bitte mindestens eine Abfragerichtung auswaehlen", Toast.LENGTH_LONG).show();
    }
}
```

11.2.13 Lösung zu Aufgabe 13

Falls Sie bei dieser Aufgabe Probleme haben sollten, lesen Sie bei früheren Lösungen zum Erstellen von View-Komponenten in Layout-Dateien nach.

11.2.14 Lösung zu Aufgabe 14

Falls Sie bei dieser Aufgabe Probleme haben sollten, lesen Sie bei früheren Lösungen zum Erstellen von View-Komponenten in Layout-Dateien nach.

11.2.15 Lösung zu Aufgabe 15

Falls Sie bei dieser Aufgabe Probleme haben sollten, lesen Sie bei früheren Lösungen zum Erstellen von View-Komponenten in Layout-Dateien nach.

11.2.16 Lösung zu Aufgabe 16

Damit die Methode `toggleButtonClicked` auch aufgerufen wird, weisen wir diese beiden `ToggleButton` in der XML-Ansicht mit dem Statement `android:onClick="toggleButtonClicked"` zu.
Bei Klicken auf einen der beiden `ToggleButton` soll der andere ausgewählt werden. Ist bisher der deutsche `ToggleButton` ausgewählt, setzen wir diesen auf `OFF` und den englischen auf `ON` und setzen zudem die Variable `deutschSelected` auf `false`. Ist bisher der englische `ToggleButton` ausgewählt verfahren wir analog genau anders rum:

```
/**
* Wird aufgerufen, wenn auf eines der beiden Switch-Objekte geklickt wird
*
* @param view
*/
public void toggleButtonClicked(View view) {
    ToggleButton deutsch = (ToggleButton) findViewById(R.id.toggledeutsch);
    ToggleButton englisch = (ToggleButton) findViewById(R.id.toggleenglisch);
    // Ist bei Klicken auf einen der beiden ToggleButton bisher der deutsche ToggleButton ausgewaehlt
    if (deutschSelected) {
        deutsch.setChecked(false);
        englisch.setChecked(true);
        deutschSelected = false;
    // Ist bei Klicken auf einen der beiden ToggleButton bisher der englische ToggleButton ausgewaehlt
    } else {
        deutsch.setChecked(true);
        englisch.setChecked(false);
        deutschSelected = true;
    }
}
```

11.2.17 Lösung zu Aufgabe 17

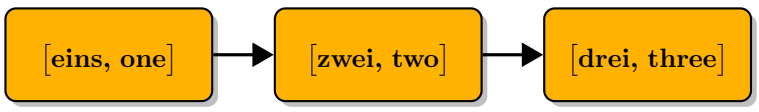
SQL-Anfrage für alle Vokabelpaare nach der deutschen Bedeutung sortiert:

```
SELECT deutsch,englisch
FROM vocTable
ORDER BY deutsch
```

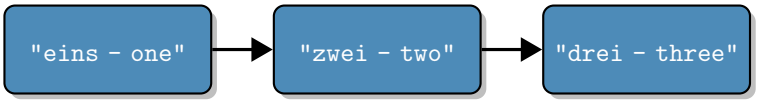
Für die englische Bedeutung analog mit `ORDER BY englisch`.

11.2.18 Lösung zu Aufgabe 18

(a) Der Cursor hat genauso viele Cursor-Elemente, wie Vokabeln in der Tabelle `vocTable` gespeichert sind. Jedes Cursor-Element hat zwei Einträge und zwar die deutsche und englische Bedeutung einer Vokabel. Für das in der Aufgabenstellung beschriebene Szenario würde das Cursor-Objekt also wie folgt aussehen:



Der Cursor wird nun schrittweise durchlaufen und die beiden Bedeutungen werden zu **einem** String zusammengefasst, mit Trennstrich dazwischen. Dieser String wird in einer `ArrayList<String>` gespeichert. Bei unserer Aufgabenstellung hat die `ArrayList<String>` nach vollständigem Durchlaufen des Cursors folgenden Aufbau:



Den Datentyp `ArrayList<String>` verwenden wir, da wir diesen dem `Spinner` anschließend direkt zuweisen können. Der `Spinner` zeigt die Einträge der `ArrayList<String>` als Dropdown-Liste an.

(b) Die Methode `nachEnglischerBedeutungSortiert()`:

```
/**
* Liste aller Vokabeln, nach deutscher Bedeutung sortiert
*
* @return
*/
public ArrayList<String> nachEnglischerBedeutungSortiert() {
    ArrayList<String> list = new ArrayList<String>();
    Cursor c = db.rawQuery("select englisch,deutsch from vocTable order by englisch;",null);

    if (c.moveToFirst()) {
       list.add(c.getString(0) + " - " + c.getString(1));

       while (c.moveToNext()) {
           list.add(c.getString(0) + " - " + c.getString(1));
       }
    }
    return list;
}
```

11.2.19 Lösung zu Aufgabe 19

Die Methode `updateSpinner()` muss in der `onCreate`-Methode und am Ende der `toggleButtonClicked`-Methode aufgerufen werden:

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    //Spinner aktualisieren
    updateSpinner();
}

public void toggleButtonClicked(View view) {
    ...
    //Spinner aktualisieren
    updateSpinner();
}
```

11.2.20 Lösung zu Aufgabe 20

Das SQL-Statement, um alle Einträge der Tabelle `vocTable` nach deutsch Bedeutung sortiert auslesen zu können, kennen wir bereits aus der Lösung von Aufgabe 17 auf Seite 127:

```
SELECT deutsch,englisch
FROM vocTable
ORDER BY deutsch
```

In der Methode `getVocByPositionDeutsch(int pos)` springen wir nun mit der Methode `moveToPosition(int pos)` an das gesuchte Element im Cursor und lesen die deutsche und englische Bedeutung der gesuchten Vokabel

aus. Diese speichern wir in einem String-Array, welches wir zurückgeben:

```
/**
 * Vokabel ueber ihre Position aus der Datenbank holen (nach deutsch sortiert)
 *
 * @param pos
 * @return
 */
public String[] getVocByPositionDeutsch(int pos) {
    String[] vokabel = new String[2];
    Cursor c = db.rawQuery("select deutsch,englisch from vocTable order by deutsch;",null);
    // Zeiger auf das Element an Position pos im Cursor setzen
    if (c.moveToPosition(pos)) {
        vokabel[0] = c.getString(0);
        vokabel[1] = c.getString(1);
    } else {
        return null;
    }
}
```

Das Vorgehen bei der Methode `getVocByPositionEnglisch(int pos)` läuft analog, nur dass wir in der SQL-Anfrage nach der englischen Bedeutung sortieren:

```
/**
 * Vokabel ueber ihre Position aus der Datenbank holen (nach englisch sortiert)
 *
 * @param pos
 * @return
 */
public String[] getVocByPositionDeutsch(int pos) {
    String[] vokabel = new String[2];
    Cursor c = db.rawQuery("select deutsch,englisch from vocTable order by englisch;",null);
    // Zeiger auf das Element an Position pos im Cursor setzen
    if (c.moveToPosition(pos)) {
        vokabel[0] = c.getString(0);
        vokabel[1] = c.getString(1);
    } else {
        return null;
    }
}
```

11.2.21 Lösung zu Aufgabe 21

(a) Folgendes SQL-Statement löscht eine Vokabel über ihre deutsche und englische Bedeutung aus der Datenbank:

```
SELECT *
FROM vocTable
ORDER BY RANDOM();
```

(b) In der Methode `deleteVoc(String deutscheBed, String englischeBed)` führen wir obiges SQL-Statement aus:

```
/**
 * Vokabel aus der Datenbank loeschen
```

```
3   *
4   * @param deutscheBed
5   * @param englischeBed
6   */
7  public void deleteVoc(String deutscheBed, String englischeBed) {
8      db.execSQL("delete from vocTable where deutsch = '" + deutscheBed+ "' and englisch = '" +
           englischeBed + "';");
9  }
```

11.2.22 Lösung zu Aufgabe 22

Falls Sie bei dieser Aufgabe Probleme haben sollten, lesen Sie bei früheren Lösungen zum Erstellen von View-Komponenten in Layout-Dateien nach.

11.2.23 Lösung zu Aufgabe 23

(a) SQL-Anfrage:

```
UPDATE vocTable
SET deutsch = 'neuDeBed', englisch = 'neuEnBed'
WHERE deutsch = 'altDeBed' AND englisch = 'altEnBed'
```

(b) Die Methode `updateVoc(String altDeBed, String altEnBed, String neuDeBed, String neuEnBed)`:

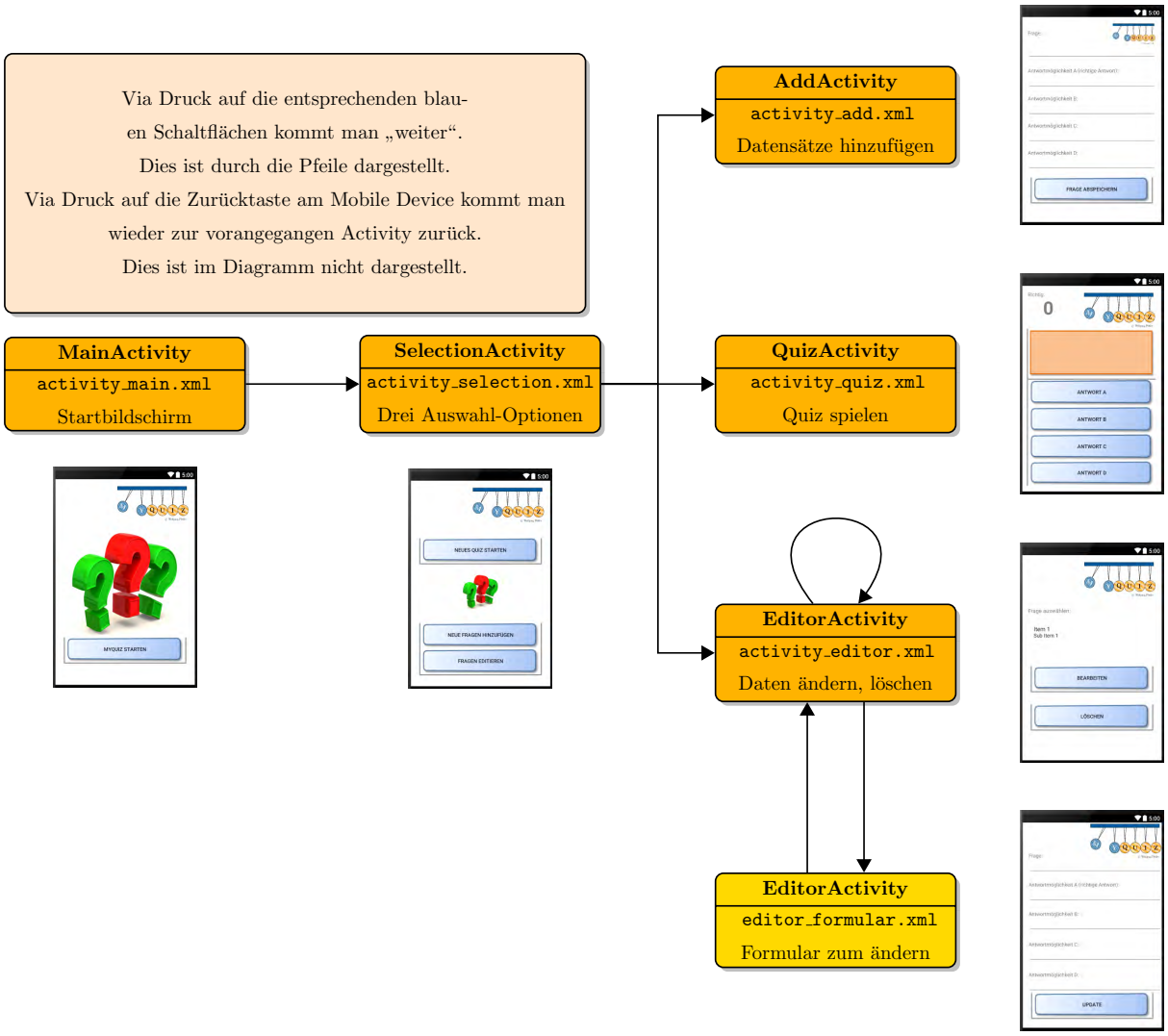
```
1  /**
2  * Ueberschreibt eine Vokabel mit neuen Bedeutungen
3  *
4  * @param altDeBed
5  * @param altEnBed
6  * @param neuDeBed
7  * @param neuEnBed
8  */
9  public void updateVoc(String altDeBed, String altEnBed, String neuDeBed,
10 String neuEnBed) {
11     db.execSQL("update vocTable set deutsch = '" + neuDeBed + "', englisch = '" + neuEnBed + "' WHERE
           deutsch = '" + altDeBed + "' and englisch = '" + altEnBed + "';");
12 }
```

11.3 MyQuiz-App

In diesem Abschnitt finden Sie einen Lösungsvorschlag zur Umsetzung der Aufgabenstellung aus Kapitel 6.

11.3.1 Zustandsdiagramm zur Benutzerführung

Ihre App könnte etwa folgenden Aufbau haben:



11.3.2 MainActivity

Das Layout der MainActivity-Klasse könnte wie folgt aussehen[1]:

Die Klasse `MainActivity.java`:

```
1  package com.example.administrator.myquiz;
2
3  import android.content.Intent;
4  import android.support.v7.app.ActionBarActivity;
5  import android.os.Bundle;
6  import android.view.Menu;
7  import android.view.MenuItem;
8  import android.view.View;
9
10 /**
11 * Created by Wolfgang Pfeffer on 31.03.2015.
12 */
13 public class MainActivity extends ActionBarActivity {
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19     }
20
21     public void startSelectionActivity(View view) {
22         Intent myIntent = new Intent(this,SelectionActivity.class);
23         startActivity(myIntent);
24     }
25 }
```

[1] Quellenangabe des verwendeten Bildes: https://fasab.files.wordpress.com/2013/08/quiz-2.jpg

11.3.3 SelectionActivity

Das Layout der SelectionActivity-Klasse könnte wie folgt aussehen[2]:

Die Klasse `SelectionActivity.java`:

```
1  package com.example.administrator.myquiz;
2
3  import android.content.Intent;
4  import android.support.v7.app.ActionBarActivity;
5  import android.os.Bundle;
6  import android.view.Menu;
7  import android.view.MenuItem;
8  import android.view.View;
9  import android.widget.Button;
10 import android.widget.TextView;
11
12 /**
13 * Created by Wolfgang Pfeffer on 31.03.2015.
14 */
15 public class SelectionActivity extends ActionBarActivity {
16
17     @Override
18     protected void onCreate(Bundle savedInstanceState) {
19         super.onCreate(savedInstanceState);
20         setContentView(R.layout.activity_selection);
21     }
22
23
24     public void startNewQuiz(View view) {
25         Intent myIntent = new Intent(this,QuizActivity.class);
26         startActivity(myIntent);
27     }
28
29
30     public void startAddActivity(View view) {
31         Intent myIntent = new Intent(this,AddActivity.class);
```

[2]Quellenangabe des verwendeten Bildes: https://fasab.files.wordpress.com/2013/08/quiz-2.jpg

```
32          startActivity(myIntent);
33      }
34
35      public void startEditorActivity(View view) {
36          Intent myIntent = new Intent(this,EditorActivity.class);
37          startActivity(myIntent);
38      }
39  }
```

11.3.4 AddActivity

Idee

- 5 Eingabefelder - 1 für die Frage, 4 für die Antwortmöglichkeiten
- die richtige Antwort muss immer als erstes eingegeben werden und wir auch immer unter `A` in der Datenbank gespeichert
- in der QuizActivity können wir so leicht überprüfen, ob der Benutzer die richtige Antwort ausgewählt hat

Das Layout der AddActivity-Klasse könnte wie folgt aussehen:

Die Klasse `AddActivity.java`:

```
1  package com.example.administrator.myquiz;
2
3  import android.support.v7.app.ActionBarActivity;
4  import android.os.Bundle;
5  import android.view.Menu;
6  import android.view.MenuItem;
7  import android.view.View;
8  import android.widget.EditText;
9  import android.widget.TextView;
```

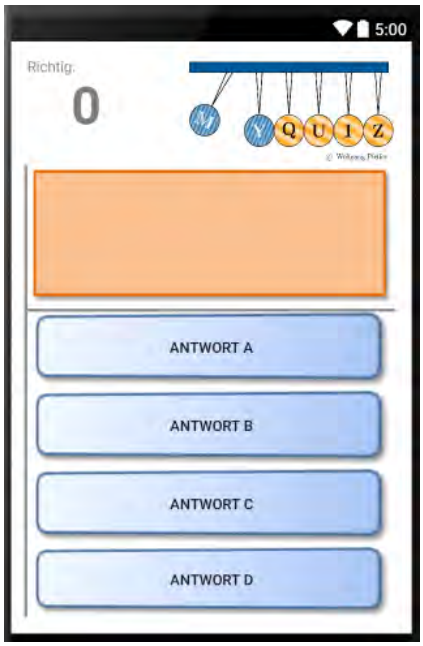
```
10 import android.widget.Toast;
11
12 /**
13 * Created by Wolfgang Pfeffer on 31.03.2015.
14 */
15 public class AddActivity extends ActionBarActivity {
16
17     private EditText frage;
18     private EditText antwortA;
19     private EditText antwortB;
20     private EditText antwortC;
21     private EditText antwortD;
22
23     private DataHandler dbHandler;
24
25     @Override
26     protected void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28         setContentView(R.layout.activity_add);
29
30         frage = (EditText) findViewById(R.id.eingabefeldfrage);
31         antwortA = (EditText) findViewById(R.id.eingabefeldA);
32         antwortB = (EditText) findViewById(R.id.eingabefeldB);
33         antwortC = (EditText) findViewById(R.id.eingabefeldC);
34         antwortD = (EditText) findViewById(R.id.eingabefeldD);
35         // Objekt der Klasse DataHandler fuer den Datenbankzugriff
36         dbHandler = new DataHandler(this);
37     }
38
39     /**
40     * Wird auf die Hinzufuegen-Schaltflaeche geklickt, wird ueberprueft,
41     * ob alle Felder ausgefuellt wurden und in diesem Fall wird der Datensatz in der Datenbank
42     * abgespeichert
43     * @param view
44     */
45     public void addButtonClicked(View view) {
46         String f = frage.getText().toString();
47         String a = antwortA.getText().toString();
48         String b = antwortB.getText().toString();
49         String c = antwortC.getText().toString();
50         String d = antwortD.getText().toString();
51
52         if (f.equals("") || a.equals("") || b.equals("") || c.equals("") || d.equals("")) {
53             Toast.makeText(this,"Bitte alle Eingabefelder ausfuellen!",Toast.LENGTH_LONG).show();
54         } else {
55             dbHandler.insertData(f,a,b,c,d);
56             Toast.makeText(this,"Datensatz abgespeichert - Fragen-Anzahl: " + dbHandler.getSize(),
57             Toast.LENGTH_LONG).show();
58         }
59     }
60 }
```

11.3.5 QuizActivity

Idee:

- Für die Antwortmöglichkeiten wurden vier Button-Objekte verwendet
- Bei Aufruf der Activity-Klasse wird ein zufälliger Datensatz aus der Datenbank geholt und angezeigt
- Wird auf eine Antwort geklickt, wird überprüft, ob die Antwort richtig war und dann entsprechend der Hintergrund des Button-Objekts auf grün bzw. rot geändert

Das Layout der QuizActivity-Klasse könnte wie folgt aussehen:



Die Klasse `QuizActivity.java`:

```
1 package com.example.administrator.myquiz;
2
```

```
3  import android.graphics.drawable.Drawable;
4  import android.support.v7.app.ActionBarActivity;
5  import android.os.Bundle;
6  import android.test.UiThreadTest;
7  import android.view.View;
8  import android.widget.Button;
9  import android.widget.TextView;
10
11 import java.util.ArrayList;
12 import java.util.Random;
13
14 /**
15 * Created by Wolfgang Pfeffer on 31.03.2015.
16 */
17 public class QuizActivity extends ActionBarActivity {
18
19     private Button buttonFrage;
20     private Button buttonA;
21     private Button buttonB;
22     private Button buttonC;
23     private Button buttonD;
24     private Button[] buttons;
25     private TextView punkte;
26
27     private DataHandler dbHandler;
28
29     private String richtigeAntwort;
30     private boolean buttonsEnabled;
31
32     @Override
33     protected void onCreate(Bundle savedInstanceState) {
34         super.onCreate(savedInstanceState);
35         setContentView(R.layout.activity_quiz);
36
37         buttonFrage = (Button) findViewById(R.id.buttonFrage);
38         buttonA = (Button) findViewById(R.id.buttontA);
39         buttonB = (Button) findViewById(R.id.buttonB);
40         buttonC = (Button) findViewById(R.id.buttonC);
41         buttonD = (Button) findViewById(R.id.buttonD);
42         buttons = new Button[] {buttonA, buttonB, buttonC, buttonD };
43         punkte = (TextView) findViewById(R.id.highscore);
44         dbHandler = new DataHandler(this);
45         buttonsEnabled = false;
46         neueFrage();
47     }
48
49
50     /**
51     * Hilfsmethode neueFrage, die einen zufaelligen Datensatz aus der Datenbank holt und die
52     * Frage bzw. die permutierten Antwortmoeglichkeiten auf den dafuer vorgesehenen
53     * View-Objekten anzeigt.
54     */
55     private void neueFrage() {
56         String[] data = dbHandler.getRandomData();
57         // die richtige Antwort wird abgespeichert
58         richtigeAntwort = data[1];
59         // die Frage wird entsprechend angezeigt
60         buttonFrage.setText(data[0]);
61         ArrayList<String> dataList = new ArrayList<String>();
62
63         // die Antwortmoeglichkeiten werden in eine ArrayList eingefuegt
64         for (int i = 1; i < 5; i++) {
65             dataList.add(data[i]);
66         }
67
```

```
68          Random rnd = new Random();
69          int i = 0;
70
71          // es werden nun zufaellig Elemente (Antwortmoeglichkeiten) aus der ArrayList genommen und
72          // diese nacheinander den vier verschiedenen Button-Objekten hinzugefuegt
73          while (!dataList.isEmpty()) {
74              int size = dataList.size();
75              int pos = rnd.nextInt(size);
76              buttons[i].setText(dataList.get(pos));
77              dataList.remove(pos);
78              i++;
79          }
80          buttonsEnabled = true;
81      }
82
83      /**
84      * Wird auf eine Antwort-Schaltflaeche geklickt, wird ueberprueft,
85      * ob die Antwort richtig war und dann entsprechend ein gruenes oder rotes Buttonbild als
86      * Hintergrund gesetzt
87      * @param view
88      */
89      public void buttonClicked(View view) {
90          if (buttonsEnabled) {
91              String antwort = ((Button) view).getText().toString();
92
93              if (antwort.equals(richtigeAntwort)) {
94                  ((Button) view).setBackground(getResources().getDrawable(R.drawable.richtig));
95                  punkte.setText((Integer.parseInt(punkte.getText().toString())+1)+"");
96              } else {
97                  ((Button) view).setBackground(getResources().getDrawable(R.drawable.falsch));
98                  punkte.setText(0+"");
99              }
100             view.invalidate();
101
102             buttonsEnabled = false;
103             buttonFrage.setText("Neue Frage anzeigen");
104         }
105     }
106
107     /**
108     * Ueber den Fragebogen kann man sich nach Beantworten einer Frage die naechste Frage
109     * anzeigen lassen
110     * @param view
111     */
112     public void fragebuttonClicked(View view) {
113         if (!buttonsEnabled) {
114             reset();
115             neueFrage();
116         }
117     }
118
119     /**
120     * Setzt die Farben der Button-Objekte zurueck
121     */
122     private void reset() {
123         for (int i = 0; i < buttons.length; i++) {
124             buttons[i].setBackground(getResources().getDrawable(R.drawable.antwort));
125         }
126     }
127 }
```

11.3.6 EditorActivity

Idee:

- Wir haben in der Datenbank jedem Datensatz eine ID-Nummer gegeben. Die fügen wir im Spinner hinzu. Soll ein Datensatz gelöscht bzw. editiert werden, extrahieren wir die ID wieder aus dem String im Spinner und löschen bzw. editieren die entsprechenden Daten
- die EditorActivity hat zwei Layout-Dateien, zwischen denen hin und her gewechselt wird, falls ein Datensatz bearbeitet werden soll

Das Layout der EditorActivity-Klasse könnte wie folgt aussehen:

Die Klasse `EditorActivity.java`:

```
1  package com.example.administrator.myquiz;
2
3  import android.os.Bundle;
4  import android.support.v7.app.ActionBarActivity;
5  import android.util.Log;
6  import android.view.View;
7  import android.widget.ArrayAdapter;
8  import android.widget.EditText;
9  import android.widget.Spinner;
10 import android.widget.Toast;
11
12 import java.util.ArrayList;
13
14 /**
15 * Created by Wolfgang Pfeffer on 31.03.2015.
16 */
17 public class EditorActivity extends ActionBarActivity {
18
19     private EditText formularFrage;
```

```
20      private EditText formularA;
21      private EditText formularB;
22      private EditText formularC;
23      private EditText formularD;
24      private String[] currentData;
25      private DataHandler dbHandler;
26      private Spinner spinner;
27
28      @Override
29      protected void onCreate(Bundle savedInstanceState) {
30          super.onCreate(savedInstanceState);
31          setContentView(R.layout.activity_editor);
32          dbHandler = new DataHandler(this);
33          // Spinner wird bei Starten der Activity aktualisiert, d.h. alle gespeicherten Datensaetze angezeigt
34          updateSpinner();
35      }
36
37      /**
38      * Hilfsmethode updateSpinner, die die Eintraege im Spinner aktualisiert
39      */
40      private void updateSpinner() {
41          spinner = (Spinner) findViewById(R.id.spinner);
42          ArrayList<String> spinnerList;
43          spinnerList = dbHandler.getData();
44          ArrayAdapter<String> spinnerAdapter = new ArrayAdapter<String>(this,
45          R.layout.spinnerlayout, spinnerList);
46          spinnerAdapter.setDropDownViewResource(R.layout.spinnerlayout);
47          spinner.setAdapter(spinnerAdapter);
48          spinner.invalidate();
49      }
50
51      /**
52      * Wird aufgerufen, wenn auf die Loeschen-Schaltflaeche geklickt wird.
53      *
54      * @param view
55      */
56      public void deleteButtonClicked(View view) {
57          if (spinner.getAdapter().getCount() > 0) {
58              int pos = spinner.getSelectedItemPosition();
59              dbHandler.deleteData(pos + 1);
60              updateSpinner();
61          } else {
62              Toast.makeText(this, "Keine Eintraege in der Datenbank", Toast.LENGTH_LONG).show();
63          }
64      }
65
66      /**
67      * WIrd aufgerufen, wenn auf die Editieren-Schaltflaeche geklickt wird
68      *
69      * @param view
70      */
71      public void editButtonClicked(View view) {
72          if (spinner.getAdapter().getCount() > 0) {
73              provideData();
74          } else {
75              Toast.makeText(this, "Keine Eintraege in der Datenbank", Toast.LENGTH_LONG).show();
76          }
77      }
78
79      /**
80      * Wird aufgerufen, wenn auf die Update-Schaltflaeche der Layout-Datei editorformular
81      * geklickt wird. Aktualisiert den ausgewaehlten Datensatz um die eingegebenen Werte
82      *
83      * @param view
84      */
```

```
85      public void updateButtonClicked(View view) {
86          String[] newData = new String[]{currentData[0], formularFrage.getText().toString(),
87          formularA.getText().toString(), formularB.getText().toString(),
88          formularC.getText().toString(), formularD.getText().toString()};
89          boolean noEmptyEntry = true;
90
91          for (int i = 0; i < newData.length; i++) {
92              if (newData[i].equals("")) {
93                  noEmptyEntry = false;
94              }
95          }
96          // Wurden alle Felder ausgefuellt
97          if (noEmptyEntry) {
98              dbHandler.updateData(newData);
99              setContentView(R.layout.activity_editor);
100             Toast.makeText(this, "Datensatz aktualisiert", Toast.LENGTH_LONG).show();
101             updateSpinner();
102         } else {
103             Toast.makeText(this, "Bitte alle Eingabefelder ausfuellen!", Toast.LENGTH_LONG).show();
104         }
105     }
106
107     /**
108     * Hilfsmethode provideData
109     * Holt den aktuell ausgewaehlten Datensatz ueber die ID aus der Tabelle und fuellt die
110     * entsprechenden Werte in die zugehoerigen EditText-Objekte der Layout-Datei editorformular
111     */
112     private void provideData() {
113         // ID wird aus dem im Spinner ausgewaehlten String extrahiert
114         String[] parts = spinner.getSelectedItem().toString().split("-");
115         currentData = dbHandler.getDataById(Integer.parseInt(parts[0]));
116         setContentView(R.layout.editorformular);
117         formularFrage = (EditText) findViewById(R.id.formularFrage);
118         formularA = (EditText) findViewById(R.id.formularA);
119         formularB = (EditText) findViewById(R.id.formularB);
120         formularC = (EditText) findViewById(R.id.formularC);
121         formularD = (EditText) findViewById(R.id.formularD);
122         formularFrage.setText(currentData[1]);
123         formularA.setText(currentData[2]);
124         formularB.setText(currentData[3]);
125         formularC.setText(currentData[4]);
126         formularD.setText(currentData[5]);
127     }
128 }
```

11.3.7 DataHandler

Die Klasse `DataHandler.java`:

```
1  package com.example.administrator.myquiz;
2
3  import android.content.Context;
4  import android.database.Cursor;
5  import android.database.sqlite.SQLiteDatabase;
6  import android.database.sqlite.SQLiteOpenHelper;
7  import android.util.Log;
8
9  import java.util.ArrayList;
10
11 /**
12 * Created by Wolfgang Pfeffer on 31.03.2015.
13 */
```

```
14 public class DataHandler {
15
16     public static final String DATABASE_NAME = "quizDB";
17     public static final int DATABASE_VERSION = 1;
18     public static final String TABLE_NAME = "quizTable";
19     public static final String ID = "id";
20     public static final String FRAGE = "frage";
21     public static final String A = "a";
22     public static final String B = "b";
23     public static final String C = "c";
24     public static final String D = "d";
25     private DataHelper dbHelper;
26     private SQLiteDatabase db;
27
28     /**
29     * Konstruktor der Klasse DataHandler
30     *
31     * @param ctx
32     */
33     public DataHandler(Context ctx) {
34         dbHelper = new DataHelper(ctx);
35     }
36
37     /**
38     * Methode insertData, die eine neue Frage samt Antwortmoeglichkeiten in die Datenbank einfuegt.
39     *
40     * @param myFrage Fragestellung
41     * @param myA     Antwortmoeglichkeit A (richtige Antwort)
42     * @param myB     Antwortmoeglichkeit B
43     * @param myC     Antwortmoeglichkeit C
44     * @param myD     Antwortmoeglichkeit D
45     */
46     public void insertData(String myFrage, String myA, String myB, String myC, String myD) {
47         String myId = "" + getNextID();
48         String sqlcommand = "insert into " + TABLE_NAME + "(" + ID + "," + FRAGE + "," + A + "," +
49         "" + B + "," + C + "," + D + ") values ('" + myId + "','" + myFrage + "'," +
50         "'" + myA + "','" + myB + "','" + myC + "','" + myD + "');";
51         openWrite();
52         db.execSQL(sqlcommand);
53         close();
54     }
55
56     /**
57     * Methode getSize, die die Anzahl der Datensaetze aus der Tabelle quizTable zurueckgibt
58     *
59     * @return Anzahl der Datensaetze
60     */
61     public int getSize() {
62         int size = -1;
63         String sqlcommand = "select count(id) from " + TABLE_NAME + ";";
64         openRead();
65         Cursor c = db.rawQuery(sqlcommand, null);
66
67         if (c.moveToFirst() && c.getString(0) != null) {
68             size = Integer.parseInt(c.getString(0));
69         }
70         close();
71         return size;
72     }
73
74     /**
75     * Private Hilfsmethode getNextID, die die ID-Nummer fuer den naechsten Eintrag zurueckgibt
76     * Es wird geschaut, wie viele Eintraege in der Datenbank sind. Zu dieser Anzahl wird 1
77     * addiert und diese Zahl zurueckgegeben.
78     *
```

```
79      * @return
80      */
81      private int getNextID() {
82          int nextId = 0;
83          String sqlcommand = "select max(id) from " + TABLE_NAME + ";";
84          openRead();
85          Cursor c = db.rawQuery(sqlcommand, null);
86
87          if (c.moveToFirst() && c.getString(0) != null) {
88              nextId = Integer.parseInt(c.getString(0));
89          }
90          close();
91          return nextId + 1;
92      }
93
94      /**
95      * Methode getRandomData, die einen zufaelligen Datensatz aus der Tabelle ausliest und
96      * zurueckgibt
97      *
98      * @return String[] mit Frage und den vier Antwortmoeglichkeiten
99      */
100     public String[] getRandomData() {
101         String[] data = new String[5];
102         String sqlcommand = "select " + FRAGE + "," + A + "," + B + "," + C + "," +
103         "" + D + " from " + TABLE_NAME + " order by random() limit 1;";
104         openRead();
105         Cursor c = db.rawQuery(sqlcommand, null);
106
107         if (c.moveToFirst()) {
108             for (int i = 0; i < 5; i++) {
109                 data[i] = c.getString(i);
110             }
111         }
112         close();
113         return data;
114     }
115
116     /**
117     * Methode getData, die die komplette Tabelle quizTable in einer ArrayList zurueckgibt. Ein
118     * Einrag in
119     * der ArrayList setzt sich aus einem String bestehend aus ID,
120     * Frage und den vier Antwortmoeglichkeiten zusammen
121     *
122     * @return ArrayList mit den gesamten Eintraegen der Tabelle quizTable
123     */
124     public ArrayList<String> getData() {
125         ArrayList<String> data = new ArrayList<String>();
126         String sqlcommand = "select * from " + TABLE_NAME + " order by " + ID + ";";
127         String datensatz = "";
128         openRead();
129         Cursor c = db.rawQuery(sqlcommand, null);
130
131         if (c.moveToFirst()) {
132             datensatz = c.getString(0) +"-";
133
134             for (int i = 1; i < 6; i++) {
135                 if (i != 5) {
136                     datensatz = datensatz + c.getString(i) + " \n ";
137                 } else {
138                     datensatz = datensatz + c.getString(i);
139                 }
140             }
141             data.add(datensatz);
142
143             while (c.moveToNext()) {
```

```
144                 datensatz = c.getString(0) +"-";
145                 for (int i = 1; i < 6; i++) {
146                     if (i != 5) {
147                         datensatz = datensatz + c.getString(i) + " \n ";
148                     } else {
149                         datensatz = datensatz + c.getString(i);
150                     }
151                 }
152                 data.add(datensatz);
153             }
154         }
155         close();
156         return data;
157     }
158 
159     /**
160     * Methode deleteData, die einen Datensatz anhand seiner ID aus der Datenbank loescht
161     *
162     * @param id ID des zu loeschenden Datensatzes
163     */
164     public void deleteData(int id) {
165         String sqlcommand = "delete from " + TABLE_NAME + " where " + ID + "='" + id + "';";
166         openWrite();
167         db.execSQL(sqlcommand);
168         close();
169     }
170 
171 
172     /**
173     * Methode getDataById, die einen Datensatz ueber die ID aus der Tabelle quizTable holt
174     *
175     * @param id ID des zu holenden Datensatzes
176     * @return Datensatz mit ID id
177     */
178     public String[] getDataById(int id) {
179         String[] data = new String[6];
180         String sqlcommand = "select * from " + TABLE_NAME + " where " + ID + "='" + id + "';";
181         openRead();
182         Cursor c = db.rawQuery(sqlcommand, null);
183         if (c.moveToFirst()) {
184             for (int i = 0; i < 6; i++) {
185                 data[i] = c.getString(i);
186             }
187         }
188         close();
189         return data;
190     }
191 
192     /**
193     * Methode updateData ersetzt einen bestehenden Datensatz mit der ID newData[0] durch neue
194     * Werte, die in dem String[] newData uebergeben werden
195     *
196     * @param newData aktuelle Werte
197     */
198     public void updateData(String[] newData) {
199         String sqlcommand = "update " + TABLE_NAME + " set " + ID + "='" + newData[0] + "', " +
200         FRAGE + "='" + newData[1] + "', " + A + "='" + newData[2] + "', " + B
201         + "='" + newData[3] + "', " + C + "='" + newData[4] + "', " + D + "='" +
202         newData[5] + "' where " + ID + "='" + newData[0] + "';";
203         openWrite();
204         db.execSQL(sqlcommand);
205         close();
206     }
207 
208     /**
```

```
209     * Fordert Schreibrechte an
210     */
211     private void openWrite() {
212         db = dbHelper.getWritableDatabase();
213     }
214 
215     /**
216     * Fordert Leserechte an
217     */
218     private void openRead() {
219         db = dbHelper.getReadableDatabase();
220     }
221 
222     /**
223     * Setzt die Zugriffsrechte zurueck
224     */
225     private void close() {
226         dbHelper.close();
227     }
228 
229     /**
230     * Innere Klasse fuer das Erstellen der Datenbank und Anlegen der Tabelle
231     */
232     private static class DataHelper extends SQLiteOpenHelper {
233 
234         public DataHelper(Context context) {
235             super(context, DATABASE_NAME, null, DATABASE_VERSION);
236         }
237 
238         @Override
239         public void onCreate(SQLiteDatabase db) {
240             String sqlcommand = "create table if not exists " + TABLE_NAME + "(" + ID + " text " +
241             "not null, " + FRAGE + " text not null, " + A + " text not null, " +
242             "" + B + " text not null, " + C + " text not null, " + D + " text not null);";
243             db.execSQL(sqlcommand);
244         }
245 
246         @Override
247             public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
248         }
249     }
250 }
```

11.4 My2048-App

11.4.1 Die Klasse SimpleGestureFilter

Code 11.4: SimpleGestureFilter.java

```
1  package com.example.administrator.my2048;
2
3  import android.app.Activity;
4  import android.view.GestureDetector.SimpleOnGestureListener;
5  import android.view.GestureDetector;
6  import android.view.MotionEvent;
7
8  /**
9  * Created by Wolfgang Pfeffer on 08.04.2015.
10 */
11 public class SimpleGestureFilter extends SimpleOnGestureListener{
12
13     public final static int SWIPE_UP    = 1;
14     public final static int SWIPE_DOWN  = 2;
15     public final static int SWIPE_LEFT  = 3;
16     public final static int SWIPE_RIGHT = 4;
17
18     public final static int MODE_TRANSPARENT = 0;
19     public final static int MODE_SOLID       = 1;
20     public final static int MODE_DYNAMIC     = 2;
21
22     private final static int ACTION_FAKE = -13; //just an unlikely number
23     private int swipe_Min_Distance = 100;
24     private int swipe_Max_Distance = 350;
25     private int swipe_Min_Velocity = 100;
26
27     private int mode      = MODE_DYNAMIC;
28     private boolean running = true;
29     private boolean tapIndicator = false;
30
31     private Activity context;
32     private GestureDetector detector;
33     private SimpleGestureListener listener;
34
35
36     public SimpleGestureFilter(Activity context,SimpleGestureListener sgl) {
37         this.context = context;
38         this.detector = new GestureDetector(context, this);
39         this.listener = sgl;
40     }
41
42
43     public void onTouchEvent(MotionEvent event){
44         if(!this.running) return;
45         boolean result = this.detector.onTouchEvent(event);
46         if(this.mode == MODE_SOLID) event.setAction(MotionEvent.ACTION_CANCEL);
47         else if (this.mode == MODE_DYNAMIC) {
48             if(event.getAction() == ACTION_FAKE) event.setAction(MotionEvent.ACTION_UP);
49             else if (result) event.setAction(MotionEvent.ACTION_CANCEL);
50             else if(this.tapIndicator) {
51                 event.setAction(MotionEvent.ACTION_DOWN);
52                 this.tapIndicator = false;
53             }
54         }
55         //else just do nothing, it's Transparent
56     }
57
58     public void setMode(int m){
```

```
59          this.mode = m;
60      }
61
62      public int getMode(){
63          return this.mode;
64      }
65
66      public void setEnabled(boolean status){
67          this.running = status;
68      }
69
70      public void setSwipeMaxDistance(int distance){
71          this.swipe_Max_Distance = distance;
72      }
73
74      public void setSwipeMinDistance(int distance){
75          this.swipe_Min_Distance = distance;
76      }
77
78      public void setSwipeMinVelocity(int distance){
79          this.swipe_Min_Velocity = distance;
80      }
81
82      public int getSwipeMaxDistance(){
83          return this.swipe_Max_Distance;
84      }
85
86      public int getSwipeMinDistance(){
87          return this.swipe_Min_Distance;
88      }
89
90      public int getSwipeMinVelocity(){
91          return this.swipe_Min_Velocity;
92      }
93
94
95      @Override
96      public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
97          final float xDistance = Math.abs(e1.getX() - e2.getX());
98          final float yDistance = Math.abs(e1.getY() - e2.getY());
99
100         if(xDistance > this.swipe_Max_Distance || yDistance > this.swipe_Max_Distance) return false;
101
102         velocityX = Math.abs(velocityX);
103         velocityY = Math.abs(velocityY);
104         boolean result = false;
105
106         if(velocityX > this.swipe_Min_Velocity && xDistance > this.swipe_Min_Distance) {
107             if(e1.getX() > e2.getX()) // right to left
108             this.listener.onSwipe(SWIPE_LEFT);
109             else
110             this.listener.onSwipe(SWIPE_RIGHT);
111
112             result = true;
113         } else if(velocityY > this.swipe_Min_Velocity && yDistance > this.swipe_Min_Distance) {
114             if(e1.getY() > e2.getY()) // bottom to up
115             this.listener.onSwipe(SWIPE_UP);
116             else
117             this.listener.onSwipe(SWIPE_DOWN);
118
119             result = true;
120         }
121         return result;
122     }
123
```

```
124     @Override
125     public boolean onSingleTapUp(MotionEvent e) {
126         this.tapIndicator = true;
127         return false;
128     }
129 
130     @Override
131     public boolean onDoubleTap(MotionEvent arg0) {
132         this.listener.onDoubleTap();;
133         return true;
134     }
135 
136     @Override
137         public boolean onDoubleTapEvent(MotionEvent arg0) {
138         return true;
139     }
140 
141     @Override
142     public boolean onSingleTapConfirmed(MotionEvent arg0) {
143         if(this.mode == MODE_DYNAMIC) {        // we owe an ACTION_UP, so we fake an
144             arg0.setAction(ACTION_FAKE);      //action which will be converted to an ACTION_UP later.
145             this.context.dispatchTouchEvent(arg0);
146         }
147 
148         return false;
149     }
150 
151 
152     static interface SimpleGestureListener{
153         void onSwipe(int direction);
154         void onDoubleTap();
155     }
156 }
```

11.4.2 Erstellen von wichtigen Hilfsmethoden

11.4.2.1 Matrix initialisieren

In dieser Methode müssen wir unser zweidimensionales Integer-Array `matrix` durchlaufen und alle Einträge auf 0 setzen. Dazu benötigen wir zwei `for`-Schleifen:

Code 11.5: Die Methode `initialisiereMatrix()`

```
1  /**
2  * Initialisiert die Matrix mit den Werten 0
3  */
4 
5  private void initialisiereMatrix() {
6      for (int i = 0; i < 4; i++) {
7          for (int j = 0; j < 4; j++) {
8          matrix[i][j] = 0;
9          }
10     }
11 }
```

11.4.2.2 Beschriftung der Button-Objekte aktualisieren

In dieser Methode müssen wir die Werte aus dem zweidimensionalen Integer-Array `matrix` den Button-Objekten als Beschriftung zuweisen. Je nach Wert müssen wir dann dem Button-Objekt noch das entsprechenden Hintergrundbild zuweisen:

Code 11.6: Die Methoden `updateView()` und `setBackgroundImage(Integer i, Button b)`

```
/**
* Hilfsmethode, die die aktuellen Werte den Button-Objekten zuweist
*/
private void updateView() {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            // Ist der Eintrag nicht 0, wird er als Beschriftung des entsprechenden Button-Objekts gesetzt
            if (matrix[i][j] != 0) {
                buttons[i][j].setText(matrix[i][j] + "");
                setBackgroundImage(matrix[i][j],buttons[i][j]);
                // ansonsten gibt es keine Beschriftung
            } else {
                buttons[i][j].setText("");
                buttons[i][j].setBackground(getResources().getDrawable(R.drawable.stufe00));
            }
        }
    }
}

/**
* Hilfsmethode zum Setzen des Hintergrundbildes
* @param i
* @param b
*/
private void setBackgroundImage(Integer i, Button b) {
    switch (i) {
        case 2:
            b.setBackground(getResources().getDrawable(R.drawable.stufe01));
            break;
        case 4:
            b.setBackground(getResources().getDrawable(R.drawable.stufe011));
            break;
        case 8:
            b.setBackground(getResources().getDrawable(R.drawable.stufe02));
            break;
        case 16:
            b.setBackground(getResources().getDrawable(R.drawable.stufe021));
            break;
        case 32:
            b.setBackground(getResources().getDrawable(R.drawable.stufe03));
            break;
        case 64:
            b.setBackground(getResources().getDrawable(R.drawable.stufe04));
            break;
        case 128:
            b.setBackground(getResources().getDrawable(R.drawable.stufe05));
            break;
        case 256:
            b.setBackground(getResources().getDrawable(R.drawable.stufe06));
            break;
        case 512:
            b.setBackground(getResources().getDrawable(R.drawable.stufe07));
            break;
        case 1024:
            b.setBackground(getResources().getDrawable(R.drawable.stufe08));
            break;
        case 2048:
            b.setBackground(getResources().getDrawable(R.drawable.stufe09));
            break;
    }
}
```

11.4.2.3 Punktestand aktualisieren

Wir erstellen zwei globale TextView-Objekte `scoreView` und `highscoreView` und eine `int`-Variable `highscore`. Die `View`-Objekte initialisieren wir in der `onCreate`-Methode (Stichwort: `findViewById`). In der Methode `updateScore` erhöhen wir den bestehenden Wert um den Wert `value` und überprüfen anschließend, ob ein neuer Highscore erzielt wurde. In diesem Fall aktualisieren wir auch die Beschriftung des TextView-Objekts `highscoreView` bzw. der Variable `highscore`:

Code 11.7: Die Methode `updateScore(int value)`

```
private TextView scoreView;
private TextView highscoreView;
private int highscore;

/**
* Hilfsmethode updateScore, die die bisher erzielten Punkte um den Wert value erhoeht
* @param value
*/
private void updateScore(int value) {
    scoreView.setText((Integer.parseInt(scoreView.getText().toString())+value)+"");
    if (Integer.parseInt(scoreView.getText().toString()) > highscore) {
        highscore = Integer.parseInt(scoreView.getText().toString());
        highscoreView.setText(highscore+"");
        dbHandler.updateHighscore(highscore);
    }
}
```

11.4.2.4 Prüfen ob noch ein Zug möglich ist

In der Methode `zugMoeglich()` suchen wir zunächst in horizontaler Richtung nach benachbarten gleichen Werten und anschließend in vertikaler Richtung. Sobald zwei benachbarte gleiche Werte gefunden werden, wird `true` zurückgegeben. Wurden keine benachbarten Werte gefunden, wird `false` zurückgegeben:

Code 11.8: Die Methode `zugMoeglich()`

```
/**
* Ueberprueft, ob noch ein Zug moeglich ist. Diese Methode wird nur aufgerufen,
* falls alle Felder belegt sind
*
* @return
*/
private boolean zugMoeglich() {
    // Zunaechst wird ueberprueft, ob es benachbarte gleiche Werte in horizontaler Richtung gibt...
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 3; j++) {
            if (matrix[i][j].equals(matrix[i][j + 1])) {
                return true;
            }
        }
    }
    // ... anschliessend wird ueberprueft, ob es benachbarte gleiche Werte in vertikaler Richtung gibt
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            if (matrix[i][j].equals(matrix[i+1][j])) {
                return true;
            }
        }
    }
    return false;
}
```

11.4.2.5 Neuen Wert an freier Stelle einfügen

In dieser Methode müssen wir zunächst zufällig ein freies Feld auswählen. Der erste Gedanke ist vielleicht, sich zwei Zahlen zwischen 0 und 3 zu erzeugen und schauen, ob das Feld leer ist. Falls nur noch wenige Felder leer sind, könnte dies allerdings einige Zeit dauern. Aus diesem Grund gehen wir zunächst das zweidimensionale Integer-Array durch und speichern uns die freien Positionen in einer `ArrayList<String>`, wobei wir die Zeilen- und Spaltennummer zu einem String zusammenfügen, wobei diese mit einem `"-"` getrennt sind (so können wir den String mit der Methode `split("-")` leicht wieder aufspalten). Wir hätten hier natürlich genauso eine `ArrayList<Integer[]>` verwenden können.
Aus dieser Liste wählen wir nun ein zufälliges Element aus und extrahieren daraus wieder die Zeilen- bzw. Spaltennummer und fügen in den entsprechenden Eintrag in unserem zweidimensionalen Integer-Array `matrix` eine 2 (zu 80%) bzw. eine 4 (zu 20%) ein.
Gab es vor dem Einfügen nur noch ein freies Feld, muss nach dem Einfügen noch geprüft werden, ob noch ein Zug möglich ist. Dazu haben wir uns bereits eine Hilfsmethode erstellt. Ist kein Zug mehr möglich, geben wir einen Toast aus und setzen `finished` auf `true`:

Code 11.9: Die Methode `neuerWert()`

```
1  private boolean finished; // wird in der onCreate-Methode auf false gesetzt
2
3  /**
4  * Hilfsmethode, die eine neue Zahl (2 oder 4) auf ein zufaellig ausgewaehltes Feld einfuegt
5  */
6  private void neuerWert() {
7      ArrayList<String> freiePositionen = new ArrayList<>();
8      // Durchlauf der Matrix und zwischenspeichern der freien Plaetze
9      for (int i = 0; i < 4; i++) {
10         for (int j = 0; j < 4; j++) {
11             if (matrix[i][j] == 0) {
12                 freiePositionen.add(i + "-" + j);
13             }
14         }
15     }
16     // Ist noch ein Platz vorhanden?
17     if (freiePositionen.size() > 0) {
18         Random rnd = new Random();
19         String rndData = freiePositionen.get(rnd.nextInt(freiePositionen.size()));
20         String[] rowColOfData = rndData.split("-");
21         int row = Integer.parseInt(rowColOfData[0]);
22         int col = Integer.parseInt(rowColOfData[1]);
23         // es wird zufaellig eine 2 oder eine 4 eingefuegt, wobei das Einfuegen einer 2
24         // wahrscheinlicher ist
25         int i = rnd.nextInt(10);
26         if (i <= 7) {
27             matrix[row][col] = 2;
28         } else {
29             matrix[row][col] = 4;
30         }
31         // war vorher nur noch ein freies Feld uebrig? Dann muss geprueft werden,
32         // ob das Spiel nun vorbei ist
33         if (freiePositionen.size() == 1) {
34             if (!zugMoeglich()) {
35                 Toast.makeText(this, "Kein␣Zug␣mehr␣moeglich", Toast.LENGTH_LONG).show();
36                 finished = true;
37             }
38         }
39         // ist kein Platz mehr vorhanden?
40     } else {
41         if (!zugMoeglich()) {
```

```
42          Toast.makeText(this, "Kein␣Zug␣mehr␣moeglich", Toast.LENGTH_LONG).show();
43          finished = true;
44          }
45      }
46  }
```

11.4.3 Vorgehensweise beim Wischen

11.4.3.1 Hilfsmethoden für das Umwandeln einer bestimmten Zeile bzw. Spalte in eine verkettete Liste

Code 11.10: Die Methoden `getRowAsList(int rownumber)` und `getColAsList(int colnumber)`

```
1   /**
2   * Hilfsmethode getRowAsList, liefert eine bestimmte Zeile der Matrix als LinkedList
3   *
4   * @param rownumber Zeilennummer, die man als Liste erhalten moechte
5   * @return entsprechende Zeile als LinkedList
6   */
7   private LinkedList<Integer> getRowAsList(int rownumber) {
8       LinkedList<Integer> list = new LinkedList<>();
9       for (int i = 0; i < 4; i++) {
10          list.add(matrix[rownumber][i]);
11      }
12      return list;
13  }
14
15  /**
16  * Hilfsmethode getColAsList, liefert eine bestimmte Spalte der Matrix als LinkedList
17  * @param colnumber Spaltennummer, die man als Liste erhalten moechte
18  * @return entsprechende Spalte als LinkedList
19  */
20  private LinkedList<Integer> getColAsList(int colnumber) {
21      LinkedList<Integer> list = new LinkedList<>();
22      for (int i = 0; i < 4; i++) {
23          list.add(matrix[i][colnumber]);
24      }
25      return list;
26  }
```

11.4.3.2 Methode, die aufgerufen wird, wenn nach links gewischt wird

In der `for`-Schleife müssen wir nun die einzelnen Elemente der Liste `list` durchlaufen. Dazu verwenden wir eine `while`-Wiederholung (`while (j < 3)`).

- **Fall 1: Es wird ein Listenelement mit Wert 0 gefunden**

  Dies bedeutet, dass in unserer Liste ein Element mit Wert 0 gefunden wird. In diesem Fall löschen wir dieses Element aus der Liste und hängen am Ende ein Element mit Wert 0 an.
  Die Variable `j` dürfen wir in diesem Fall nicht erhöhen.
  Nun müssen wir noch den Fall beachten, dass alle Elemente in der Liste den Wert 0 haben. Bisher würden wir unendlich oft einen Eintrag löschen und am Ende wieder neu hinzufügen. Aus diesem Grund erstellen wir eine Zählvariable `countEmpty`, die wir erhöhen, falls ein Element mit Wert 0 gefunden und gelöscht wird. Wurden mehr als 4 Elemente oder mehr gelöscht, brechen wir die `while`-Wiederholung mit `break` ab:

Code 11.11: Die Methode `boolean swipeLeft()`

```
/**
* Wird aufgerufen, wenn nach links gewischt wird
* @return
*/
private boolean swipeLeft() {
    Integer[][] copy = copyOfMatrix();
    LinkedList<Integer> list;
    int countempty = 0;

    for (int i = 0; i < 4; i++) {
        list = getRowAsList(i);
        countyempty = 0;
        j = 0;

        while (j < 3) {
            // wurde ein Listenelement mit Wert 0 gefunden
            if (list.get(j) == 0) {
                list.remove(j);
                list.addLast(0);
                countempty++;
                if (countempty >= 3) {
                    break;
                }
            } else {
                // TODO: Falls kein leeren Eintrag gefunden wird
            }
        }
    }
    return Arrays.deepEquals(copy,matrix);
}
```

- **Fall 2: Es wird ein Listenelement mit Wert ungleich 0 gefunden**

  - **Fall 2.1: Das direkt benachbarte Listenelement hat den gleichen Wert**

    In diesem Fall verdoppeln wir den Wert dieses Listenelements und löschen den benachbarten Eintrag aus der Liste. Dafür fügen wir am Ende der Liste ein neues Element mit Wert 0 ein.
    Die Variable `j` erhöhen wir um 1:

Code 11.12: Die Methode `boolean swipeLeft()`

```
private boolean swipeLeft() {
    Integer[][] copy = copyOfMatrix();
    LinkedList<Integer> list;
    int countempty = 0;

    for (int i = 0; i < 4; i++) {
        list = getRowAsList(i);
        countyempty = 0;
        j = 0;

        while (j < 3) {
            // wurde ein Listenelement mit Wert 0 gefunden
            if (list.get(j) == 0) {
                list.remove(j);
                list.addLast(0);
                countempty++;
                if (countempty >= 3) {
                    break;
                }
            } else {
```

```
21                  // Listenelement mit Wert ungleich 0 und benachbarten Listenelement, das den
                        gleichen Wert hat
22                  if (list.get(j).equals(list.get(j + 1))) {
23                      list.set(j, list.get(j) * 2);
24                      list.remove(j + 1);
25                      list.addLast(0);
26                      j++;
27                  }
28              }
29          }
30      }
31      return Arrays.deepEquals(copy,matrix);
32  }
```

– **Fall 2.2: Das direkt benachbarte Listenelement hat den Wert 0 und das übernächste Listenelement hat den gleichen Wert (nur möglich, falls j < 2)**

In diesem Fall verdoppeln wir den Wert dieses Listenelements, und löschen die beiden nachfolgenden Listenelemente (beachten Sie, dass dies nicht durch `list.remove(i+1)` und anschließend `list.remove(i+2)` funktioniert!!). Anschließend fügen wir am Ende der Liste zwei neue Elemente mit Wert 0 ein. Nach dieser Aktion können keine weiteren Aktionen mehr durchgeführt werden und wir beenden die `while`-Wiederholung mit `break`:

Code 11.13: Die Methode `boolean swipeLeft()`

```
1   private boolean swipeLeft() {
2       Integer[][] copy = copyOfMatrix();
3       LinkedList<Integer> list;
4       int countempty = 0;
5
6       for (int i = 0; i < 4; i++) {
7           list = getRowAsList(i);
8           countyempty = 0;
9           j = 0;
10
11          while (j < 3) {
12              // wurde ein Listenelement mit Wert 0 gefunden
13              if (list.get(j) == 0) {
14                  list.remove(j);
15                  list.addLast(0);
16                  countempty++;
17                  if (countempty >= 3) {
18                      break;
19                  }
20              } else {
21                  // Listenelement mit Wert ungleich 0 und benachbarten Listenelement, das den
                        gleichen Wert hat
22                  if (list.get(j).equals(list.get(j + 1))) {
23                      list.set(j, list.get(j) * 2);
24                      list.remove(j + 1);
25                      list.addLast(0);
26                      j++;
27                  } else if ((j < 2) && list.get(j + 1) == 0 && list.get(j).equals(list.get(j + 2))
                        ) {
28                      list.set(j, list.get(j) * 2);
29                      list.remove(j + 1);
30                      list.remove(j + 1);
31                      list.addLast(0);
32                      list.addLast(0);
33                      break;
```

```
34                  }
35              }
36          }
37      }
38      return Arrays.deepEquals(copy,matrix);
39  }
```

– **Fall 2.3: Das nächste und übernächste Listenelement hat den Wert 0 und das über-übernächste Listenelement hat den gleichen Wert (nur möglich, falls $j = 0$)**

In diesem Fall verdoppeln wir den Wert dieses Listenelements, löschen das letzte Element aus der Liste (das Element mit dem gleichen Wert) und fügen ein Element mit Wert 0 am Ende der Liste ein. Nach dieser Aktion können keine weiteren Aktionen mehr durchgeführt werden. Folglich beenden wir die `while`-Wiederholung mit `break`:

Code 11.14: Die Methode `boolean swipeLeft()`

```
1   private boolean swipeLeft() {
2       Integer[][] copy = copyOfMatrix();
3       LinkedList<Integer> list;
4       int countempty = 0;
5
6       for (int i = 0; i < 4; i++) {
7           list = getRowAsList(i);
8           countyempty = 0;
9           j = 0;
10
11          while (j < 3) {
12              // wurde ein Listenelement mit Wert 0 gefunden
13              if (list.get(j) == 0) {
14                  list.remove(j);
15                  list.addLast(0);
16                  countempty++;
17                  if (countempty >= 3) {
18                      break;
19                  }
20              } else {
21                  // Listenelement mit Wert ungleich 0 und benachbarten Listenelement, das den
                        gleichen Wert hat
22                  if (list.get(j).equals(list.get(j + 1))) {
23                      list.set(j, list.get(j) * 2);
24                      list.remove(j + 1);
25                      list.addLast(0);
26                      j++;
27                      // Listenelement mit Wert ungleich 0 kann mit dem uebernachsten Listenelement
                            zusammengefasst werden
28                  } else if (j < 2 && list.get(j + 1) == 0 && list.get(j).equals(list.get(j + 2)))
                        {
29                      list.set(j, list.get(j) * 2);
30                      list.remove(j + 1);
31                      list.remove(j + 1);
32                      list.addLast(0);
33                      list.addLast(0);
34                      break;
35                      // Listenelement mit Wert ungleich 0 kann mit dem ueberuebernachsten
                            Listenelement zusammengefasst werden
36                  } else if (j == 0 && list.get(j + 1) == 0 && list.get(j + 2) == 0 && list.get(j +
                        3).equals(list.get(j))) {
37                      list.set(j, list.get(j) * 2);
38                      list.removeLast();
39                      list.addLast(0);
```

```
                    break;
                }
            }
        }
    }
    return Arrays.deepEquals(copy,matrix);
}
```

– **Fall 2.4: Das Listenelement kann mit keinem anderen Listenelement zusammengefasst werden**

In diesem Fall wird die Variable `j` einfach um 1 erhöht:

Code 11.15: Die Methode `boolean swipeLeft()`

```
private boolean swipeLeft() {
    Integer[][] copy = copyOfMatrix();
    LinkedList<Integer> list;
    int countempty = 0;

    for (int i = 0; i < 4; i++) {
        list = getRowAsList(i);
        countyempty = 0;
        j = 0;

        while (j < 3) {
            // wurde ein Listenelement mit Wert 0 gefunden
            if (list.get(j) == 0) {
                list.remove(j);
                list.addLast(0);
                countempty++;
                if (countempty >= 3) {
                    break;
                }
            } else {
                // Listenelement mit Wert ungleich 0 und benachbarten Listenelement, das den gleichen Wert hat
                if (list.get(j).equals(list.get(j + 1))) {
                    list.set(j, list.get(j) * 2);
                    list.remove(j + 1);
                    list.addLast(0);
                    j++;
                    // Listenelement mit Wert ungleich 0 kann mit dem uebernachsten Listenelement zusammengefasst werden
                } else if (j < 2 && list.get(j + 1) == 0 && list.get(j).equals(list.get(j + 2))) {
                    list.set(j, list.get(j) * 2);
                    list.remove(j + 1);
                    list.remove(j + 1);
                    list.addLast(0);
                    list.addLast(0);
                    break;
                } else if (j == 0 && list.get(j + 1) == 0 && list.get(j + 2) == 0 && list.get(j + 3).equals(list.get(j))) {
                    list.set(j, list.get(j) * 2);
                    list.removeLast();
                    list.addLast(0);
                    break;
                } else {
                    j++;
                }
            }
```

```
44          }
45      }
46      return Arrays.deepEquals(copy,matrix);
47  }
```

Nach den durchgeführten Listenoperationen muss die bearbeitete Zeile noch in unserem zweidimensionalen Integer-Array `matrix` aktualisiert werden. Wir erstellen uns hierzu eine Hilfsmethode `updateRowInMatrix(rownumber)`:

Code 11.16: Die Methode `boolean swipeLeft()`

```
1  private void updateRowInMatrix(LinkedList<Integer> row, int rownumber) {
2      for (int j = 0; j < 4; j++) {
3          matrix[rownumber][j] = row.get(j);
4      }
5  }
```

Diese Methode rufen wir an geeigneter Stelle (bevor die bearbeitete Liste überschrieben wird) auf:

Code 11.17: Die Methode `boolean swipeLeft()`

```
1  private boolean swipeLeft() {
2      Integer[][] copy = copyOfMatrix();
3      LinkedList<Integer> list;
4      int countempty = 0;
5
6      for (int i = 0; i < 4; i++) {
7          list = getRowAsList(i);
8          countyempty = 0;
9          j = 0;
10
11         while (j < 3) {
12             // wurde ein Listenelement mit Wert 0 gefunden
13             if (list.get(j) == 0) {
14                 list.remove(j);
15                 list.addLast(0);
16                 countempty++;
17                 if (countempty >= 3) {
18                     break;
19                 }
20             } else {
21                 // Listenelement mit Wert ungleich 0 und benachbarten Listenelement, das den gleichen
                       Wert hat
22                 if (list.get(j).equals(list.get(j + 1))) {
23                     list.set(j, list.get(j) * 2);
24                     list.remove(j + 1);
25                     list.addLast(0);
26                     j++;
27                 } else if (j < 2 && list.get(j + 1) == 0 && list.get(j).equals(list.get(j + 2))) {
28                     list.set(j, list.get(j) * 2);
29                     list.remove(j + 1);
30                     list.remove(j + 1);
31                     list.addLast(0);
32                     list.addLast(0);
33                     break;
34                 } else if (j == 0 && list.get(j + 1) == 0 && list.get(j + 2) == 0 && list.get(j + 3).
                       equals(list.get(j))) {
35                     list.set(j, list.get(j) * 2);
36                     list.removeLast();
37                     list.addLast(0);
38                     break;
39                 } else {
40                     j++;
41                 }
```

```
42              }
43          }
44          // bearbeitete Zeile im zweidimensionalen Integer-Array matrix aktualisieren
45          updateRowInMatrix(list,i);
46      }
47      return Arrays.deepEquals(copy,matrix);
48  }
```

11.4.3.3 Methoden, die aufgerufen werden, wenn nach rechts, oben bzw. unten gewischt wird

Die Methode `swipeRight()` ist sehr ähnlich zu der Methode `swipeLeft()`, nur dass wir dieses Mal die Liste von rechts nach links durchlaufen müssen:

Code 11.18: Die Methode `boolean swipeRight()`

```
1   private boolean swipeRight() {
2       int j = 3;
3       int countempty = 0;
4       Integer[][] copy = copyOfMatrix();
5       LinkedList<Integer> list;
6       for (int i = 0; i < 4; i++) {
7           countempty = 0;
8           j = 3;
9           list = getRowAsList(i);
10          while (j > 0) {
11              if (list.get(j) == 0) {
12                  list.remove(j);
13                  list.addFirst(0);
14                  countempty++;
15                  if (countempty >= 4) {
16                      break;
17                  }
18              } else {
19                  if (list.get(j).equals(list.get(j - 1))) {
20                      updateScore(list.get(j-1)*2);
21                      list.set(j, list.get(j - 1) * 2);
22                      list.remove(j - 1);
23                      list.addFirst(0);
24                      j--;
25                  } else if (j >= 2 && list.get(j).equals(list.get(j - 2)) && list.get(j - 1) == 0) {
26                      updateScore(list.get(j-2)*2);
27                      list.set(j, list.get(j - 2) * 2);
28                      list.remove(j - 2);
29                      list.remove(j - 2);
30                      list.addFirst(0);
31                      list.addFirst(0);
32                      break;
33                  } else if (j == 3 && list.get(j).equals(list.get(j - 3)) && list.get(j - 1) == 0 && list.get
                        (j - 2) == 0) {
34                      updateScore(list.get(j-3)*2);
35                      list.set(j, list.get(j - 3) * 2);
36                      list.removeFirst();
37                      list.addFirst(0);
38                      break;
39                  } else {
40                      j--;
41                  }
42              }
43          }
44          updateRowInMatrix(list, i);
45      }
46      return Arrays.deepEquals(copy, matrix);
```

```
47 }
```

Die Fälle „nach oben wischen“ bzw. „nach unten wischen“ lassen sich auf die Fälle „nach links wischen“ bzw. „nach rechts wischen“ zurückführen. Wir müssen in den beiden Fällen nur die vier Spalten der Matrix in Listen umwandeln. Folglich erstellen wir uns die Methoden `LinkedList<Integer> getColAsList(int colnumber)` und `void updateColInMatrix(LinkedList<Integer> col, int colnumber)`:

Code 11.19: Die Methode `LinkedList<Integer> getColAsList(int colnumber)` und `void updateColInMatrix(LinkedList<Integer> col, int colnumber)`

```
1  /**
2  * Hilfsmethode getColAsList, liefert eine bestimmte Spalte der Matrix als LinkedList
3  * @param colnumber Spaltennummer, die man als Liste erhalten moechte
4  * @return entsprechende Spalte als LinkedList
5  */
6  private LinkedList<Integer> getColAsList(int colnumber) {
7      LinkedList<Integer> list = new LinkedList<>();
8      for (int i = 0; i < 4; i++) {
9          list.add(matrix[i][colnumber]);
10     }
11     return list;
12 }
13
14 private void updateColInMatrix(LinkedList<Integer> col, int colnumber) {
15     for (int j = 0; j < 4; j++) {
16         matrix[j][colnumber] = col.get(j);
17     }
18 }
```

Um die Fälle „nach oben wischen“ bzw. „nach unten wischen“ in die beiden Methoden `swipeLeft()` bzw. `swipeRight()` integrieren zu können, benötigen wir noch einen Aufrufparameter, mit dem wir unterscheiden können, ob nach links oder oben bzw. nach rechts oder unten gewischt wurde. Wir haben uns dazu entschieden, hierfür einen `enum Direction` zu verwenden:

Code 11.20: Der `enum Direction`

```
1  package com.example.administrator.my2048;
2
3  /**
4  * Created by Administrator on 12.04.2015.
5  */
6  public enum Direction {
7      UP,DOWN,RIGHT,LEFT;
8  }
```

Wir haben die beiden Methoden `swipeLeft()` bzw. `swipeRight()` in `swipeLeftOrUp(Direction dir)` bzw. `swipeRightOrDown(Direction dir)` umbenannt und wie folgt erweitert:

Code 11.21: Der `enum Direction`

```
1  /**
2  * Hilfsmethode, die aufgerufen wird, wenn nach links oder oben gewischt wird
```

```
*/
private boolean swipeLeftOrUp(Direction dir) {
    int j = 0;
    int countempty = 0;
    Integer[][] copy = copyOfMatrix();
    LinkedList<Integer> list;
    for (int i = 0; i < 4; i++) {
        countempty = 0;
        j = 0;
        if (dir == Direction.UP) {
            list = getColAsList(i);
        } else {
            list = getRowAsList(i);
        }
        while (j < 3) {
            if (list.get(j) == 0) {
                list.remove(j);
                list.addLast(0);
                countempty++;
                if (countempty > 4) {
                    break;
                }
            } else {
                if (list.get(j).equals(list.get(j + 1))) {
                    updateScore(list.get(j+1)*2);
                    list.set(j, list.get(j + 1) * 2);
                    list.remove(j + 1);
                    list.addLast(0);
                    j++;
                } else if ((j < 2) && list.get(j + 1) == 0 && list.get(j).equals(list.get(j + 2))) {
                    updateScore(list.get(j+2)*2);
                    list.set(j, list.get(j + 2) * 2);
                    list.remove(j + 1);
                    list.remove(j + 1);
                    list.addLast(0);
                    list.addLast(0);
                    break;
                } else if (j == 0 && list.get(j + 1) == 0 && list.get(j + 2) == 0 && list.get(3).equals(list
                     .get(j))) {
                    updateScore(list.get(3)*2);
                    list.set(j, list.get(3) * 2);
                    list.removeLast();
                    list.addLast(0);
                    break;
                } else {
                    j++;
                }
            }
        }
        if (dir == Direction.UP) {
            updateColInMatrix(list, i);
        } else {
            updateRowInMatrix(list, i);
        }
    }
    return Arrays.deepEquals(copy, matrix);
}

/**
* Hilfsmethode, die aufgerufen wird, wenn nach rechts oder unten gewischt wird
*/
private boolean swipeRightOrDown(Direction dir) {
    int j = 3;
    int countempty = 0;
    Integer[][] copy = copyOfMatrix();
```

```
67      LinkedList<Integer> list;
68      for (int i = 0; i < 4; i++) {
69          countempty = 0;
70          j = 3;
71          if (dir == Direction.DOWN) {
72              list = getColAsList(i);
73          } else {
74              list = getRowAsList(i);
75          }
76          while (j > 0) {
77              if (list.get(j) == 0) {
78                  list.remove(j);
79                  list.addFirst(0);
80                  countempty++;
81                  if (countempty >= 4) {
82                      break;
83                  }
84              } else {
85                  if (list.get(j).equals(list.get(j - 1))) {
86                      updateScore(list.get(j-1)*2);
87                      list.set(j, list.get(j - 1) * 2);
88                      list.remove(j - 1);
89                      list.addFirst(0);
90                      j--;
91                  } else if (j >= 2 && list.get(j).equals(list.get(j - 2)) && list.get(j - 1) == 0) {
92                      updateScore(list.get(j-2)*2);
93                      list.set(j, list.get(j - 2) * 2);
94                      list.remove(j - 2);
95                      list.remove(j - 2);
96                      list.addFirst(0);
97                      list.addFirst(0);
98                      break;
99                  } else if (j == 3 && list.get(j).equals(list.get(j - 3)) && list.get(j - 1) == 0 && list.get
                        (j - 2) == 0) {
100                     updateScore(list.get(j-3)*2);
101                     list.set(j, list.get(j - 3) * 2);
102                     list.removeFirst();
103                     list.addFirst(0);
104                     break;
105                 } else {
106                     j--;
107                 }
108             }
109         }
110         if (dir == Direction.DOWN) {
111             updateColInMatrix(list, i);
112         } else {
113             updateRowInMatrix(list, i);
114         }
115     }
116     return Arrays.deepEquals(copy, matrix);
117 }
```

11.4.4 Score aktualisieren

Immer wenn zwei Felder zu einem neuen zusammengefügt werden, rufen wir die Methode `updateScore(int value)` auf, etwa:

Code 11.22: Aufrufen der Methode `updateScore(int value)`

```
1 ...
2     if (list.get(j).equals(list.get(j + 1))) {
```

```
3           updateScore(list.get(j+1)*2);
4           list.set(j, list.get(j + 1) * 2);
5           list.remove(j + 1);
6           list.addLast(0);
7           j++;
8       }
9   ...
```

11.4.5 Persistentes Speichern der Daten

Um Daten persistent speichern zu können, benötigen wir eine Datenbankverbindung. Wir erstellen uns hierzu, wie bereits aus den vorangegangenen Aufgaben gewohnt, eine Klasse `DataHandler`:

Code 11.23: Die Klasse `DataHandler`

```
1   package com.example.administrator.my2048;
2
3   import android.content.Context;
4   import android.database.Cursor;
5   import android.database.sqlite.SQLiteDatabase;
6   import android.database.sqlite.SQLiteOpenHelper;
7
8   /**
9   * Created by Administrator on 13.04.2015.
10  */
11  public class DataHandler {
12
13      /*
14      * Attribute fuer Datenbanknamen und Datenbankversion
15      */
16      public static final String DATABASE_NAME = "gameDB";
17      public static final int DATABASE_VERSION = 1;
18
19      /*
20      * Attribute fuer Tabellen- und Spaltennamen
21      */
22      public static final String TABLE_NAME = "gameTable";
23      public static final String ZEILE_EINS = "erstezeile";
24      public static final String ZEILE_ZWEI = "zweitezeile";
25      public static final String ZEILE_DREI = "drittezeile";
26      public static final String ZEILE_VIER = "viertezeile";
27      public static final String HIGHSCORE = "highscore";
28
29      /*
30      * Attribute fuer Referenz auf die Datenbank, sowie ein Objekt der Klasse DataBaseHelper
31      */
32      private SQLiteDatabase db;
33      private DataBaseHelper dbHelper;
34
35      /**
36      * Konstruktor
37      * @param ctx
38      */
39      public DataHandler(Context ctx) {
40          dbHelper = new DataBaseHelper(ctx);
41      }
42
43      /**
44      * Hilfsmethode openWrite() zum Anfordern von Schreibrechten
45      */
46      private void openWrite() {
47          db = dbHelper.getWritableDatabase();
```

```
48      }
49
50      /**
51      * Hilfsmethode openRead() zum Anfordern von Leserechten
52      */
53      private void openRead() {
54          db = dbHelper.getReadableDatabase();
55      }
56
57      private void close() {
58          dbHelper.close();
59      }
60
61      /**
62      * Methode saveData. Bekommt als Aufrufparameter ein zweidimensionales Integer-Array,
63      * sowie den Highscore und speichert diesen in die Datenbank. Das Integer-Array wird dabei
64      * zeilenweise in einem String konvertiert, die Eintraege dabei mit einem - getrennt
65      * @param matrix
66      * @param highscore
67      */
68      public void saveData(Integer[][] matrix, int highscore) {
69          String[] rows = new String[4];
70          String hs = highscore + "";
71          for (int i = 0; i < 4; i++) {
72              String token = "";
73              for (int j = 0; j < 4; j++) {
74                  if (j < 3) {
75                      token = token + matrix[i][j] + "-";
76                  } else {
77                  token = token + matrix[i][j];
78                  }
79              }
80              rows[i] = token;
81          }
82
83          openWrite();
84          String sql = "delete from " + TABLE_NAME;
85          db.execSQL(sql);
86          sql = "insert into " + TABLE_NAME + " values('" + rows[0] + "','" + rows[1] + "','" +
87          rows[2] + "','" + rows[3] + "','" + hs + "');";
88          db.execSQL(sql);
89          close();
90      }
91
92      /**
93      * Methode getData(). Liest die Eintraege aus der Tabelle gameTable aus und konvertiert die
94      * Strings zurueck in ein zweidimensionales Integer-Array
95      * @return zweidimensionales Integer-Array
96      */
97      public Integer[][] getData() {
98          Integer[][] matrix = null;
99          String[][] rows = new String[4][];
100         openRead();
101         String sql = "select " + ZEILE_EINS + "," + ZEILE_ZWEI + "," + ZEILE_DREI + "," +
102         "" + ZEILE_VIER + " from " + TABLE_NAME + ";";
103         Cursor c = db.rawQuery(sql,null);
104         if (c.moveToFirst()) {
105             matrix = new Integer[4][4];
106             for (int i = 0; i < 4; i++) {
107                 String s = c.getString(0);
108                 String[] a = c.getString(0).split("-");
109                 rows[i] = c.getString(i).split("-");
110             }
111             for (int i = 0; i < 4; i++) {
112                 for (int j = 0; j < 4; j++) {
```

```
113                     matrix[i][j] = Integer.parseInt(rows[i][j]);
114                 }
115             }
116         }
117         return matrix;
118     }
119 
120     /**
121     * Methode updateHighscore(int highscore). Aktualisiert den Eintrag in der Spalte highscore
122     * der Tabelle gameTable auf den Wert highscore
123     * @param highscore
124     */
125     public void updateHighscore(int highscore) {
126         String hs = highscore + "";
127         String sql = "update " + TABLE_NAME + " set " + HIGHSCORE + "=" + hs + ";";
128         openWrite();
129         db.execSQL(sql);
130         close();
131     }
132 
133     /**
134     * Methode getHighscore(). Liest den in der Spalte highscore der Tabelle gameTable
135     * gespeicherten Wert aus und gibt diesen zurueck
136     * @return
137     */
138     public int getHighscore() {
139         int highscore = -1;
140         String sql = "select " + HIGHSCORE + " from " + TABLE_NAME + ";";
141         openRead();
142         Cursor c = db.rawQuery(sql,null);
143         if (c.moveToFirst()) {
144             highscore = Integer.parseInt(c.getString(0));
145         }
146         return highscore;
147     }
148 
149 
150     /**
151     * Innere Klasse DataBaseHelper fuer das Erstellen und Verwalten der Datenbank
152     */
153     private static class DataBaseHelper extends SQLiteOpenHelper {
154 
155         /**
156         * Konstruktor
157         * @param ctx
158         */
159         public DataBaseHelper(Context ctx) {
160             super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
161         }
162 
163         @Override
164         public void onCreate(SQLiteDatabase db) {
165             String sql = "create table if not exists " + TABLE_NAME + "(" + ZEILE_EINS + " text " +
166             "not null, " + ZEILE_ZWEI + " text not null, " + ZEILE_DREI + " text not " +
167             "null, " + ZEILE_VIER + " text not null, " + HIGHSCORE + " text not null);";
168             db.execSQL(sql);
169 
170         }
171 
172         @Override
173         public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
174 
175         }
176     }
177 }
```

Wir erstellen in der Activity-Klasse die Methode `onPause()`, in der wir die aktuellen Daten abspeichern:

Code 11.24: Die Methode `onPause()`

```
@Override
protected void onPause() {
    super.onPause();
    dbHandler.saveData(matrix,highscore);
}
```

In der `onCreate()`-Methode überprüfen wir, ob Daten in der Tabelle gespeichert sind. In diesem Fall laden wir diese:

Code 11.25: Die Methode `onCreate()`

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    matrix = dbHandler.getData();
    if (matrix == null) {
        matrix = new Integer[4][4];
        initialisiereMatrix();
        neuerWert();
    }
    updateView();
    dbHandler.saveData(matrix,highscore);
}
```

Denken Sie auch daran, den Highscore abzuspeichern, falls ein neuer Highscore erzielt wurde bzw. die Daten zu sichern, wenn ein neues Spiel gestartet wird.

11.4.6 Die komplette Klasse `GameActivity`

Code 11.26: Die Methode `onCreate()`

```
package com.example.administrator.my2048;

import android.app.Activity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.example.administrator.my2048.SimpleGestureFilter.SimpleGestureListener;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Random;

/**
```

```
19 * Created by Wolfgang Pfeffer on 08.04.2015.
20 */
21 public class GameActivity extends Activity implements SimpleGestureListener {
22
23     private SimpleGestureFilter detector;
24     private Integer[][] matrix;
25     private LinkedList<Integer> rows;
26     private LinkedList<Integer> cols;
27     private Button[][] buttons;
28     private boolean finished;
29     private TextView scoreView;
30     private TextView highscoreView;
31     private DataHandler dbHandler;
32     private int highscore;
33
34     @Override
35     protected void onCreate(Bundle savedInstanceState) {
36         super.onCreate(savedInstanceState);
37         setContentView(R.layout.activity_game);
38         dbHandler = new DataHandler(this);
39         highscore = dbHandler.getHighscore();
40         highscoreView = (TextView) findViewById(R.id.highscore);
41         highscoreView.setText(highscore+"");
42         scoreView = (TextView) findViewById(R.id.score);
43         detector = new SimpleGestureFilter(this, this);
44
45         Button[][] viewbuttons = {{(Button) findViewById(R.id.b11), (Button) findViewById(R.id.b12),
46                 (Button) findViewById(R.id.b13), (Button) findViewById(R.id.b14)},
47             {(Button) findViewById(R.id.b21), (Button) findViewById(R.id.b22),
48                 (Button) findViewById(R.id.b23), (Button) findViewById(R.id.b24)},
49             {(Button) findViewById(R.id.b31), (Button) findViewById(R.id.b32),
50                 (Button) findViewById(R.id.b33), (Button) findViewById(R.id.b34)},
51             {(Button) findViewById(R.id.b41), (Button) findViewById(R.id.b42),
52                 (Button) findViewById(R.id.b43), (Button) findViewById(R.id.b44)}};
53         buttons = viewbuttons;
54         finished = false;
55         matrix = dbHandler.getData();
56         if (matrix == null) {
57             matrix = new Integer[4][4];
58             initialisiereMatrix();
59             neuerWert();
60         }
61         updateView();
62         dbHandler.saveData(matrix,highscore);
63     }
64
65     @Override
66     protected void onPause() {
67         super.onPause();
68         dbHandler.saveData(matrix,highscore);
69     }
70
71
72
73     @Override
74     public boolean dispatchTouchEvent(MotionEvent me) {
75         this.detector.onTouchEvent(me);
76         return super.dispatchTouchEvent(me);
77     }
78
79
80     @Override
81     public void onSwipe(int direction) {
82         if (!finished) {
83             switch (direction) {
```

```
84                  case SimpleGestureFilter.SWIPE_RIGHT:
85                  if (!swipeRightOrDown(Direction.RIGHT)) {
86                      neuerWert();
87                      updateView();
88                  }
89                  break;
90                  case SimpleGestureFilter.SWIPE_LEFT:
91                  if (!swipeLeftOrUp(Direction.LEFT)) {
92                      neuerWert();
93                      updateView();
94                  }
95                  break;
96                  case SimpleGestureFilter.SWIPE_UP:
97                  if (!swipeLeftOrUp(Direction.UP)) {
98                      neuerWert();
99                      updateView();
100                 }
101                 break;
102                 case SimpleGestureFilter.SWIPE_DOWN:
103                 if (!swipeRightOrDown(Direction.DOWN)) {
104                     neuerWert();
105                     updateView();
106                 }
107                 break;
108             }
109         }
110     }
111
112     @Override
113     public void onDoubleTap() {
114     }
115
116     /**
117     * Hilfsmethode getRowAsList, liefert eine bestimmte Zeile der Matrix als LinkedList
118     *
119     * @param rownumber Zeilennummer, die man als Liste erhalten moechte
120     * @return entsprechende Zeile als LinkedList
121     */
122     private LinkedList<Integer> getRowAsList(int rownumber) {
123         LinkedList<Integer> list = new LinkedList<>();
124         for (int i = 0; i < 4; i++) {
125             list.add(matrix[rownumber][i]);
126         }
127         return list;
128     }
129
130     /**
131     * Hilfsmethode getColAsList, liefert eine bestimmte Spalte der Matrix als LinkedList
132     * @param colnumber Spaltennummer, die man als Liste erhalten moechte
133     * @return entsprechende Spalte als LinkedList
134     */
135     private LinkedList<Integer> getColAsList(int colnumber) {
136         LinkedList<Integer> list = new LinkedList<>();
137         for (int i = 0; i < 4; i++) {
138             list.add(matrix[i][colnumber]);
139         }
140         return list;
141     }
142
143     /**
144     * Initialisiert die Matrix mit den Werten 0
145     */
146
147     private void initialisiereMatrix() {
148         for (int i = 0; i < 4; i++) {
```

```
149             for (int j = 0; j < 4; j++) {
150                 matrix[i][j] = 0;
151             }
152         }
153     }
154 
155     /**
156     * Hilfsmethode, die eine neue Zahl (2 oder 4) auf ein zufaellig ausgewaehltes Feld einfuegt
157     */
158     private void neuerWert() {
159         ArrayList<String> freiePositionen = new ArrayList<>();
160         // Durchlauf der Matrix und zwischenspeichern der freien Plaetze
161         for (int i = 0; i < 4; i++) {
162             for (int j = 0; j < 4; j++) {
163                 if (matrix[i][j] == 0) {
164                     freiePositionen.add(i + "-" + j);
165                 }
166             }
167         }
168         // Ist noch ein Platz vorhanden?
169         if (freiePositionen.size() > 0) {
170             Random rnd = new Random();
171             String rndData = freiePositionen.get(rnd.nextInt(freiePositionen.size()));
172             String[] rowColOfData = rndData.split("-");
173             int row = Integer.parseInt(rowColOfData[0]);
174             int col = Integer.parseInt(rowColOfData[1]);
175             // es wird zufaellig eine 2 oder eine 4 eingefuegt, wobei das Einfuegen einer 2
176             // wahrscheinlicher ist
177             int i = rnd.nextInt(10);
178             if (i <= 7) {
179                 matrix[row][col] = 2;
180             } else {
181             matrix[row][col] = 4;
182         }
183         // war vorher nur noch ein freies Feld uebrig? Dann muss geprueft werden,
184         // ob das Spiel nun vorbei ist
185         if (freiePositionen.size() == 1) {
186             if (!zugMoeglich()) {
187                 Toast.makeText(this, "Kein␣Zug␣mehr␣moeglich", Toast.LENGTH_LONG).show();
188                 finished = true;
189             }
190         }
191         // ist kein Platz mehr vorhanden?
192     } else {
193         if (!zugMoeglich()) {
194             Toast.makeText(this, "Kein␣Zug␣mehr␣moeglich", Toast.LENGTH_LONG).show();
195             finished = true;
196         }
197     }
198 }
199 
200     /**
201     * Hilfsmethode, die die aktuellen Werte den Button-Objekten zuweist
202     */
203     private void updateView() {
204         for (int i = 0; i < 4; i++) {
205             for (int j = 0; j < 4; j++) {
206                 if (matrix[i][j] != 0) {
207                     buttons[i][j].setText(matrix[i][j] + "");
208                     setBackgroundImage(matrix[i][j],buttons[i][j]);
209                 } else {
210                     buttons[i][j].setText("");
211                     buttons[i][j].setBackground(getResources().getDrawable(R.drawable.stufe00));
212                 }
213             }
```

```
214         }
215     }
216 
217     /**
218     * Hilfsmethode zum Setzen des Hintergrundbildes
219     * @param i
220     * @param b
221     */
222     private void setBackgroundImage(Integer i, Button b) {
223         switch (i) {
224             case 2:
225             b.setBackground(getResources().getDrawable(R.drawable.stufe01));
226             break;
227             case 4:
228             b.setBackground(getResources().getDrawable(R.drawable.stufe011));
229             break;
230             case 8:
231             b.setBackground(getResources().getDrawable(R.drawable.stufe02));
232             break;
233             case 16:
234             b.setBackground(getResources().getDrawable(R.drawable.stufe021));
235             break;
236             case 32:
237             b.setBackground(getResources().getDrawable(R.drawable.stufe03));
238             break;
239             case 64:
240             b.setBackground(getResources().getDrawable(R.drawable.stufe04));
241             break;
242             case 128:
243             b.setBackground(getResources().getDrawable(R.drawable.stufe05));
244             break;
245             case 256:
246             b.setBackground(getResources().getDrawable(R.drawable.stufe06));
247             break;
248             case 512:
249             b.setBackground(getResources().getDrawable(R.drawable.stufe07));
250             break;
251             case 1024:
252             b.setBackground(getResources().getDrawable(R.drawable.stufe08));
253             break;
254             case 2048:
255             b.setBackground(getResources().getDrawable(R.drawable.stufe09));
256             break;
257         }
258     }
259 
260     /**
261     * Ueberprueft, ob noch ein Zug moeglich ist. Diese Methode wird nur aufgerufen,
262     * falls alle Felder belegt sind
263     *
264     * @return
265     */
266     private boolean zugMoeglich() {
267         for (int i = 0; i < 4; i++) {
268             for (int j = 0; j < 3; j++) {
269                 if (matrix[i][j].equals(matrix[i][j + 1])) {
270                     return true;
271                 }
272             }
273         }
274         for (int i = 0; i < 3; i++) {
275             for (int j = 0; j < 4; j++) {
276                 if (matrix[i][j].equals(matrix[i+1][j])) {
277                     return true;
278                 }
```

```
279             }
280         }
281         return false;
282     }
283 
284     /**
285     * Wird aufgerufen, wenn nach links gewischt wird
286     * @return
287     */
288     private boolean swipeLeft() {
289         Integer[][] copy = copyOfMatrix();
290         LinkedList<Integer> list;
291 
292         for (int i = 0; i < 4; i++) {
293             list = getRowAsList(i);
294             // TODO Operationen durchfuehren und matrix aktualisieren
295         }
296 
297         return Arrays.deepEquals(copy,matrix);
298     }
299 
300     /**
301     * Erstellt eine echte Kopie des zweidimensionalen Integer-Arrays matrix
302     * @return
303     */
304     private Integer[][] copyOfMatrix() {
305         Integer[][] copy = new Integer[matrix.length][];
306         for (int i = 0; i < 4; i++) {
307             copy[i] = matrix[i].clone();
308         }
309         return copy;
310     }
311 
312 
313     /**
314     * Hilfsmethode, die aufgerufen wird, wenn nach links oder oben gewischt wird
315     */
316     private boolean swipeLeftOrUp(Direction dir) {
317         int j = 0;
318         int countempty = 0;
319         Integer[][] copy = copyOfMatrix();
320         LinkedList<Integer> list;
321         for (int i = 0; i < 4; i++) {
322             countempty = 0;
323             j = 0;
324             if (dir == Direction.UP) {
325                 list = getColAsList(i);
326             } else {
327             list = getRowAsList(i);
328             }
329             while (j < 3) {
330                 if (list.get(j).equals(0)) {
331                     list.remove(j);
332                     list.addLast(0);
333                     countempty++;
334                     if (countempty > 4) {
335                         break;
336                     }
337                 } else {
338                     if (list.get(j).equals(list.get(j + 1))) {
339                         updateScore(list.get(j+1)*2);
340                         list.set(j, list.get(j + 1) * 2);
341                         list.remove(j + 1);
342                         list.addLast(0);
343                         j++;
```

```
344                     } else if ((j < 2) && list.get(j + 1).equals(0) && list.get(j).equals(list.get(j + 2)))
                            {
345                         updateScore(list.get(j+2)*2);
346                         list.set(j, list.get(j + 2) * 2);
347                         list.remove(j + 1);
348                         list.remove(j + 1);
349                         list.addLast(0);
350                         list.addLast(0);
351                         break;
352                     } else if (j == 0 && list.get(j + 1).equals(0) && list.get(j + 2).equals(0) && list.get
                            (3).equals(list.get(j))) {
353                         updateScore(list.get(3)*2);
354                         list.set(j, list.get(3) * 2);
355                         list.removeLast();
356                         list.addLast(0);
357                         break;
358                     } else {
359                         j++;
360                     }
361                 }
362             }
363             if (dir == Direction.UP) {
364                 updateColInMatrix(list, i);
365             } else {
366                 updateRowInMatrix(list, i);
367             }
368         }
369         return Arrays.deepEquals(copy, matrix);
370     }
371 
372     private boolean swipeRightOrDown(Direction dir) {
373         int j = 3;
374         int countempty = 0;
375         Integer[][] copy = copyOfMatrix();
376         LinkedList<Integer> list;
377         for (int i = 0; i < 4; i++) {
378             countempty = 0;
379             j = 3;
380             if (dir == Direction.DOWN) {
381                 list = getColAsList(i);
382             } else {
383             list = getRowAsList(i);
384             }
385             while (j > 0) {
386                 if (list.get(j).equals(0)) {
387                     list.remove(j);
388                     list.addFirst(0);
389                     countempty++;
390                     if (countempty >= 4) {
391                         break;
392                     }
393                 } else {
394                     if (list.get(j).equals(list.get(j - 1))) {
395                         updateScore(list.get(j-1)*2);
396                         list.set(j, list.get(j - 1) * 2);
397                         list.remove(j - 1);
398                         list.addFirst(0);
399                         j--;
400                     } else if (j >= 2 && list.get(j).equals(list.get(j - 2)) && list.get(j - 1).equals(0)) {
401                         updateScore(list.get(j-2)*2);
402                         list.set(j, list.get(j - 2) * 2);
403                         list.remove(j - 2);
404                         list.remove(j - 2);
405                         list.addFirst(0);
406                         list.addFirst(0);
```

```
407                         break;
408                     } else if (j == 3 && list.get(j).equals(list.get(j - 3)) && list.get(j - 1).equals(0) &&
                               list.get(j - 2).equals(0)) {
409                         updateScore(list.get(j-3)*2);
410                         list.set(j, list.get(j - 3) * 2);
411                         list.removeFirst();
412                         list.addFirst(0);
413                         break;
414                     } else {
415                         j--;
416                     }
417                 }
418             }
419             if (dir == Direction.DOWN) {
420                 updateColInMatrix(list, i);
421             } else {
422                 updateRowInMatrix(list, i);
423             }
424         }
425         return Arrays.deepEquals(copy, matrix);
426     }
427 
428     /**
429     * Hilfsmethode updateScore, die die bisher erzielten Punkte um den Wert value erhoeht
430     * @param value
431     */
432     private void updateScore(int value) {
433         scoreView.setText((Integer.parseInt(scoreView.getText().toString())+value)+"");
434         if (Integer.parseInt(scoreView.getText().toString()) > highscore) {
435             highscore = Integer.parseInt(scoreView.getText().toString());
436             highscoreView.setText(highscore+"");
437             dbHandler.updateHighscore(highscore);
438         }
439     }
440 
441 
442 
443     private void updateRowInMatrix(LinkedList<Integer> row, int rownumber) {
444         for (int j = 0; j < 4; j++) {
445             matrix[rownumber][j] = row.get(j);
446         }
447     }
448 
449     /**
450     *
451     * @param col
452     * @param colnumber
453     */
454     private void updateColInMatrix(LinkedList<Integer> col, int colnumber) {
455         for (int j = 0; j < 4; j++) {
456             matrix[j][colnumber] = col.get(j);
457         }
458     }
459 
460 
461     public void neuesSpiel(View view) {
462         scoreView.setText("0");
463         initialisiereMatrix();
464         neuerWert();
465         updateView();
466         dbHandler.saveData(matrix,highscore);
467     }
468 }
```

Kapitel 12

Quellennachweise

Englisch-Vokabel-App

(1) Logo der App:

http://images.myposter.de/motive/england-flagge-bilder_motiv.jpg

(2) Glühbirne in den Aufgabenstellungen:

http://gluehbirne.ist.org/images/gluehbirne/500px-Dialog-information_on.svg.png

(3) SQL-Logo:

http://www.ufukcatalca.com/wp-content/uploads/SQL-logo.jpg

(4) Deutschland-Flagge in der VocTestActivity-Klasse:

http://de.forwallpaper.com/wallpaper/german-flag-germany-the-flag-colors-colors-166554.html

(5) England-Flagge in der VocTestActivity-Klasse:

http://pichost.me/1831813/

(6) Würfelbecher:

http://picture.yatego.com/images/3ff557d10d2888.9/wuerfelbecher-kqh/wrfelbecher--poker-zubehr.jpg

(7) Würfel-Bilder:

http://www.horoskop-orakel.de/pics/wuerfel-1.jpg

(8) Lebenszyklus Android-App:

http://developer-blog.net/wp-content/uploads/2013/04/android_lifecycle.png

(9) Datenbank-Symbol:

http://www.krug-markus.de/wp-content/uploads/2011/05/Database-2-256x256.png

(10) Laptop Client Server Kommunikation:

http://www.clipartbest.com/cliparts/aiq/b8B/aiqb8B99T.jpeg

(11) HTC Handy Client Server Kommunikation:

http://business.chip.de/bii/9/0/3/6/3/8/9/5b5917189c36c8ff.jpg

(12) Wlan-Symbol:

http://explicatus.de/wp-content/uploads/wlan-logo-explicatus.jpg

(13) EV3-Symbol:

http://guidelive.pegasus.imgix.net/img/photos/2014/07/10/GyroBoy.jpg

(14) Tuba-Logo:

https://www.wordans.com/wvc-1310776330/wordansfiles/images/2011/7/15/89022/89022_340.jpg

Kapitel 13

Haftung für Links zu Webseiten

In diesem Mdoul sind Querverweise (Links) zu Webinhalten fremder Betreiber angegeben. Auf die Inhalte dieser Webseiten haben wir keinen Einfluss.

Bei der erstmaligen Angabe haben wir den fremden Inhalt daraufhin überprüft, ob durch ihn eine mögliche zivilrechtliche oder strafrechtliche Verantwortlichkeit ausgelöst wird.
Rechtswidrige Inhalte waren zum Zeitpunkt der Verlinkung nicht erkennbar.

Trotz sorgfältiger inhaltlicher Kontrolle übernehmen wir keine Haftung für die Inhalte externer Links.
Für den Inhalt der verlinkten Seiten sind ausschließlich deren Betreiber verantwortlich.

Passau, im August 2016

Wolfgang Pfeffer