

# Android

## Übungsbeispiele



Dr. Heinz Schiffermüller  
HTBLA-Kaindorf  
Abteilung EDVO

Erstellung: August 2019  
Letzte Überarbeitung: August 2019

---

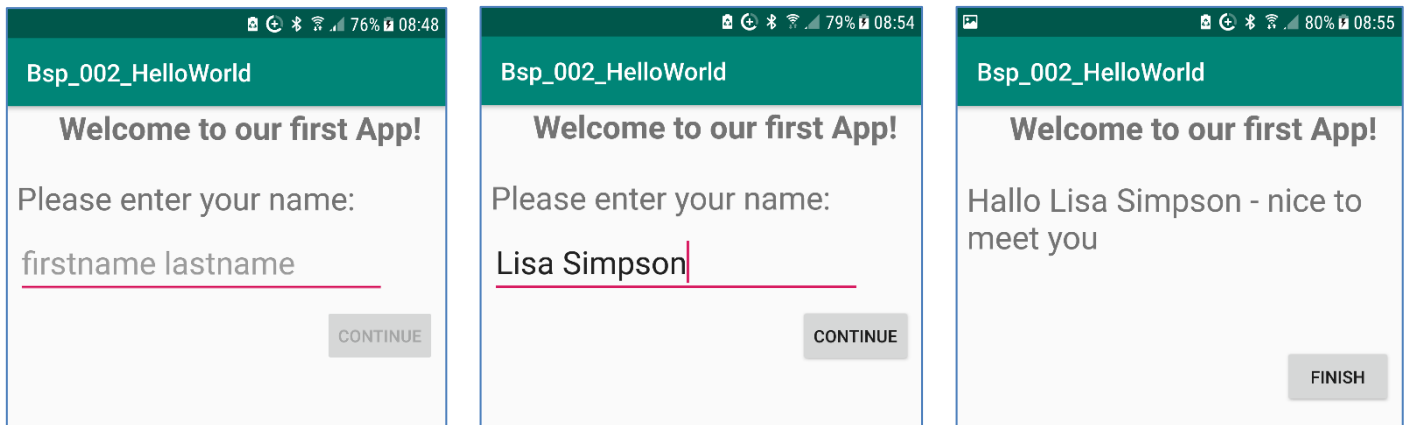
**Exa\_101\_HelloAndroid**

Erstelle eine einfache Standard App in Android Studio mit dem Begrüßungstext "Hello Android". Starte die App auf deinem Smartphone und im Emulator.

---

**Exa\_102\_HelloWorld**

HelloWorld-App bestehend aus einem Begrüßungstext, einem Eingabe-Textfeld und einem Button. Durch Klicken auf den Button wird das Textfeld ausgelesen, der Begrüßungstext wird geändert, das Eingabetextfeld verschwindet und die Beschriftung des Buttons wird von Continue auf Finish geändert:




*Kompetenzen:* LinearLayout, TextView, EditText, Button, onClick-Event

## Exa\_103\_CurrencyConverter

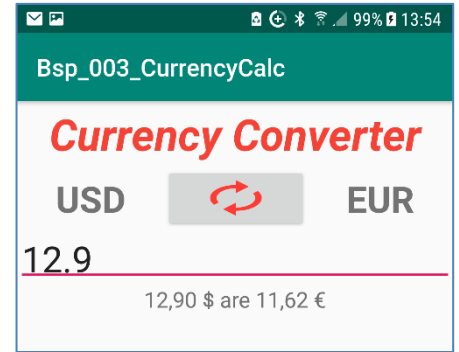
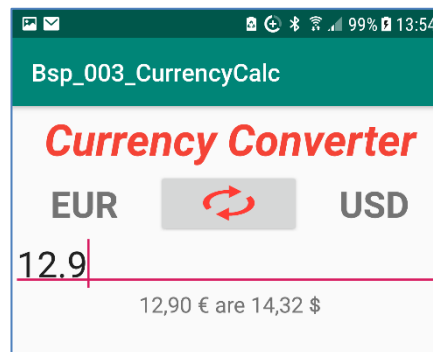
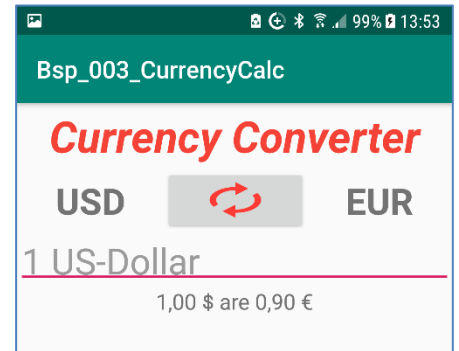
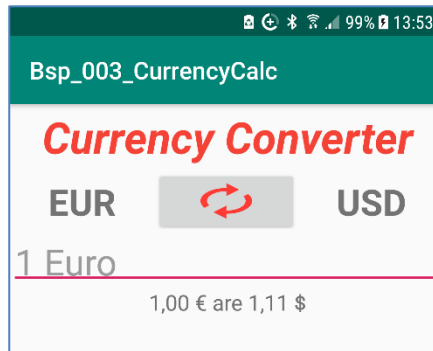


Erstelle eine App zur Umwandlung von Euro-Beträgen in Dollar und umgekehrt. Verwende dazu folgendes Layout:

**Programmablauf:**

Der Währungs-Wechsel-Button  ändert die Umrechnung von Euro in Dollar und umgekehrt. Es werden die Beschriftungen der TextViews (EUR und USD) ausgetauscht, und die aktuell angezeigte Umrechnung wird neu berechnet.



Im EditText können Beträge ganzzahlig oder als Kommazahlen eingegeben werden. Nach jeder Eingabeänderung wird die Umrechnung in der unteren TextView sofort aktualisiert. Wird der Inhalt im EditText vollständig gelöscht, so wird als Hint entweder 1 Euro oder 1 US-Dollar angezeigt.

**Programmbeschreibung:****Datei activity\_main.xml:**


Verwende ein geschachteltes **LinearLayout** um die GUI entsprechend der Abbildung zu erstellen. Für das Wechseln der Währung wird statt eines Buttons ein ImageButton verwendet.

Klasse **MainActivity.java** sind folgende Methoden zu implementieren:



**onChangeCurrency()**

-  onClick-Event für den ImageButton: Jeder Button-Klick bewirkt das Austauschen der Währungen. Zu ändern sind die Beschriftungen aller betroffenen TextView-Elemente.
-  Registriere das Event über die XML-Datei.

**afterTextChanged()**

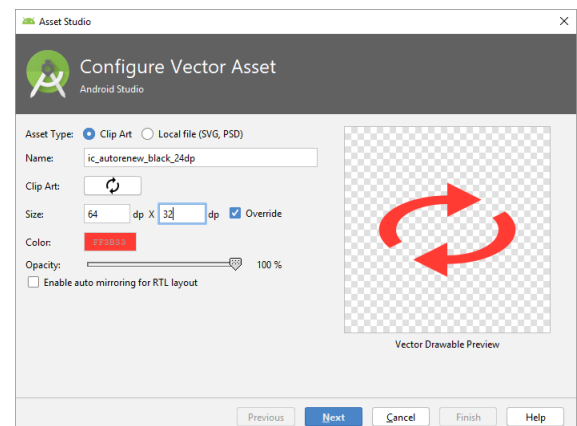
-  Implementiere das **TextWatcher**-Interface als innere, anonyme Klasse und registriere sie als **TextChangedListener** für das EditText-Element.

**convertAndUpdate()**

-  Implementiere in dieser Methode die Programm-Logik.
-  Verwende die Methode in den beiden Events.

**Hinweis:**

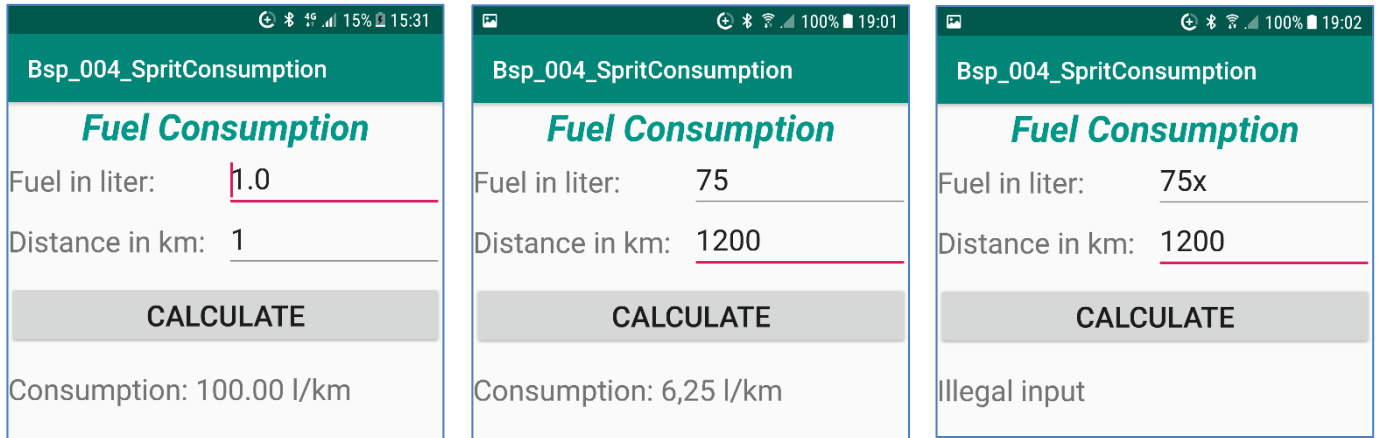
Um das Image für den Button einzufügen: im Project-View von Android Studio das **res/drawable** Verzeichnis öffnen und anklicken. Über das Kontext-Menü **New** → **Vector Asset** das Asset Studio öffnen. Auf das Icon neben **Clipart** klicken und als Suchstring **autorenew** eingeben. Das Clipart auswählen und konfigurieren. Verwende anstelle eines **Button** einen **ImageButton**.



**Kompetenzen:** LinearLayout, TextView, EditText, Button, onClick-Event

**Exa\_104\_FuelConsumption**

Erstelle eine App zur Berechnung des durchschnittlichen Benzinverbrauchs auf Basis der verbrauchten Menge an Benzin in Liter und der gefahrenen Entfernung in Kilometer. Verwende dazu folgendes Layout:

**Programmablauf:**

Die Anzeige bei Programmstart ist der ersten Abbildung zu entnehmen.

In den EditText Eingabefeldern werden der Wert für den Benzinverbrauch als Kommazahl und der Wert für die gefahrenen Kilometer als ganze Zahl eingegeben. Nach Klicken auf den Calculate-Button wird der durchschnittliche Verbrauch pro 100 km berechnet und in der unteren TextView ausgegeben.

**Programmbeschreibung:**

Datei `activity_main.xml`:

Verwende ein `RelativeLayout` um die GUI entsprechend der Abbildung zu erstellen.

Klasse `MainActivity.java` sind folgende Methoden zu implementieren:

`onClick()`

- 📌 onClick-Event für den Button: implementiere das `OnClickListener`-Interface als innere, anonyme Klasse und registriere sie für den Button.

`onCalcConsumption()`

- 📌 Methode in der die Programm-Logik implementiert wird.
- 📌 Verwende Exception-Handling bei fehlerhafter Eingabe.

**Erweiterung:**

Verwende eine grüne Schrift für die Ausgabe wenn der Verbrauch unter 10 Liter liegt, sonst eine rote Schrift.

*Kompetenzen:* RelativeLayout, Views, onClick-Event

**Exa\_105\_PocketCalculator**

☆☆

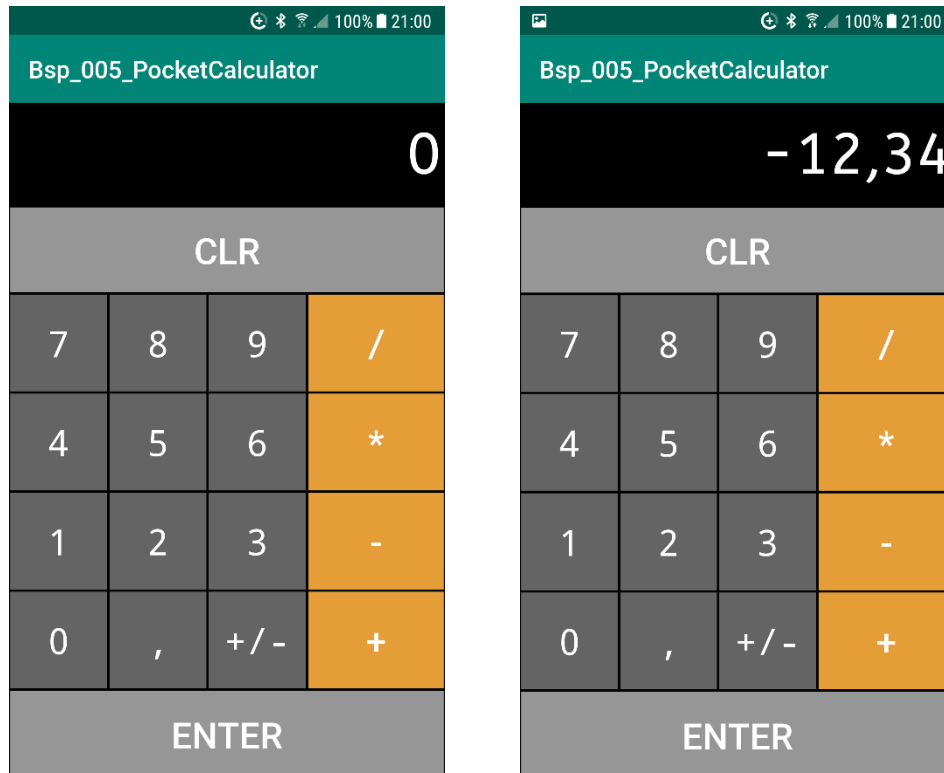
Erstelle eine App für einen einfachen Taschenrechner mit Postfix Notation. Bei der Postfix-Notation werden zuerst die beiden Werte, getrennt durch die ENTER-Taste, eingegeben und anschließend erst der Operator:

Tastenfolge Infix-Notation: **1 + 1 =**

Tastenfolge Postfix-Notation: **1 ENTER 1 +**

Der Vorteil der Postfix Notation ist, dass keine Klammerungen notwendig sind.

Verwende für den Taschenrechner folgendes Layout:

**Programmablauf:**

Bei Programmstart entspricht die Anzeige der linken Abbildung. Die CLR Taste löscht alle Eingaben – auf der Anzeige erscheint wieder die Null. Die ENTER-Taste dient zum Trennen zweier Zahlen. Die Komma-Taste (',') zur Eingabe eines Dezimaltrennzeichens – natürlich ist nur eines pro Zahl möglich. Die Plus-Minus-Taste ('+/-') fügt am Beginn der Zahl ein negatives Vorzeichen ein oder löscht es wieder. Die Operator-Tasten dienen zum Berechnen der Werte. Zu Beginn einer Zahl, darf keine Null ('0') stehen. Die Anzeige am Display ist auf maximal 12 Ziffern zu begrenzen.




**Programmbeschreibung:**

Datei `activity_main.xml`:

Verwende ein `TableLayout` um die GUI entsprechend der Abbildung zu erstellen.

Klasse `MainActivity.java` sind folgende Methoden zu implementieren:

- ☐ `onDigitClick()`: onClick-Event für alle Ziffern-Buttons (0-9). Registriere die Methode für alle Ziffern über die XML-Datei. Dient zur Anzeige aller Ziffern (0-9) auf dem Display.
- ☐ `onClear()`: Löschen der aktuellen Anzeige (das Display zeigt 0 an) und Löschen des Stacks. Registriere die Methode über eine innere anonyme Klasse.
- ☐ `onKomma()`: Einfügen des Dezimaltrennzeichens (wenn noch nicht vorhanden!). Registriere die Methode über eine innere anonyme Klasse.

-  **onSign ()**: Toggle-Button für negatives Vorzeichen (-). Registriere die Methode über eine innere anonyme Klasse.
-  **onOperator ()**: Handler-Methode für alle Operatoren (+,-,\*,/). Berechnung vom Ergebnis und Anzeige am Display. Die beiden Operanden werden vom Stack gelesen, das Ergebnis wird wieder auf dem Stack gespeichert. Registriere die Methode für alle vier Buttons über eine innere Klasse.
-  **onEnter ()**: Einlesen der Eingabe vom Display. Der eingegebene Wert bleibt am Display sichtbar und wird auf einen Stack gelegt. Anschliessend kann eine neue Zahl eingegeben werden. Registriere die Methode über eine innere anonyme Klasse.

Die Anzeige der Zahl am Display erfolgt mit einem String. Die Handlermethoden **onDigit()**, **onKomma()**, **onClear()** und **onSign()** ändern den Anzeige-String. In den Methoden **onOperator()** und **onEnter()** erfolgt die Umwandlung vom Anzeige-String in einen numerischen double-Wert.

Es sind alle Exceptions abzufangen und mit Fehlermeldungen auszugeben. Mögliche Exceptions treten auf bei Division durch Null, oder durch illegale Stack Zugriffe, z.B. wenn bei leerem Stack eine Operator Taste gedrückt wird. Verwende für die Ausgabe der Fehlermeldung einen Toast.

Die Zwischenspeicherung von eingegebenen Zahlen oder von Zwischenergebnissen erfolgt auf einem Stack. Bei Programmstart wird 0 im Display angezeigt.

Die Klasse **Stack** entsprechend dem Klassendiagramm:

**push ()**: legt einen neuen Wert auf den Stack.

**pop ()**: Löscht einen Wert vom Stack und gibt ihn zurück.

**isFull ()**: **true** wenn der Stack voll ist.

**isEmpty ()**: **true** wenn der Stack leer ist.

**clear ()**: löscht den Stack.

Exceptions sind zu werfen wenn auf einen vollen Stack geschrieben wird (Stack overflow), oder von einem leeren Stack gelesen wird.

Stack
- werte : double[10] - top : int
+ push(wert : double) : void + pop() : double + isFull() : boolean + isEmpty() : boolean + clear() : void

*Kompetenzen:* Table-Layout, Views, Event-Handling, Stack, Postfix-Notation

**Exa\_106\_TextFieldFormatter**

Erstelle eine App zur Formatierung eines mehrzeiligen Eingabe-Textfeldes. Verwende dazu folgendes Layout:

**Programmablauf:**

Im Eingabe-Textfeld können bis zu 5 Zeilen angezeigt werden. Über die beiden Checkboxes kann die Schrift fett und kursiv angezeigt werden.

Über die Seekbar kann die Schriftgröße des Textes in einem Bereich zwischen 12 und 36 dp geändert werden.

Über die drei Radiobuttons kann eine von drei unterschiedlichen Schriftarten für den Text ausgewählt werden.

Alle Änderungen der Einstellungen werden sofort auf das Eingabetextfeld angewendet.

**Programmbeschreibung:****Datei `activity_main.xml`:**

Verwende ein **RelativeLayout** um die GUI entsprechend der Abbildung zu erstellen.

Klasse **MainActivity.java** sind folgende Methoden zu implementieren:

**`onChangeFontStyle()`**

- 📱 onClick-Event für die beiden Checkboxes: Implementiere den **OnClickListener** über eine innere Klasse.

**`onChangeFontSize()`**

- 📱 Implementiere den **OnSeekBarChangeListener** als innere, anonyme Klasse und rufe die Handlermethode in der Methode **onProgressChanged** auf. Der aktuelle Wert der Schriftgröße wird auf der GUI angezeigt.

**Hinweis:** Den aktuellen Wert der Seekbar erhält man über den Parameter **progress**.

**`onChangeFont()`**

- 📱 Implementiere den **OnCheckedChangeListener** als innere, anonyme Klasse für die RadioGroup.

**Hinweis:** Die Schriftarten sind im **res/font** Verzeichnis über xml-Dateien anzulegen.

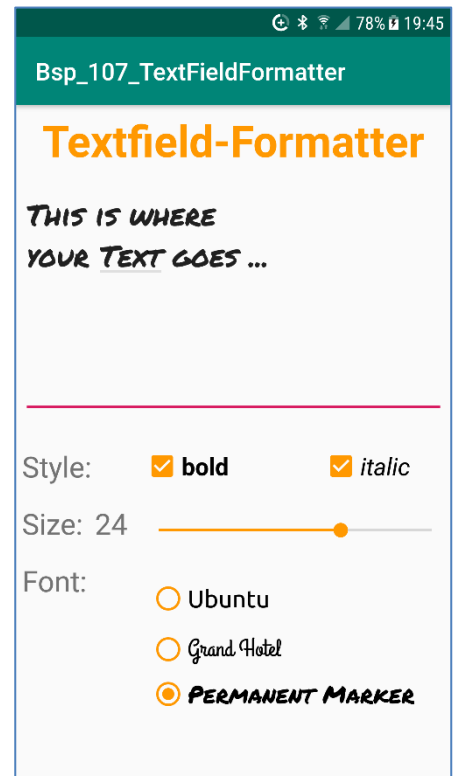
**Hinweis:**

Die Font-Eigenschaften werden über ein **Typeface**-Objekt definiert:

```
Typeface font;
font = Typeface.defaultFromStyle(Typeface.ITALIC)
font = ResourcesCompat.getFont(getApplicationContext(), R.font.ubuntu);
```

Zum Setzen der Font wird die **setTypeface()**-Methode der EditText-View verwendet.

*Kompetenzen:* Layout, Views, Event-Handling



**Exa\_107\_Quiz**

Erstelle eine App in der fünf Quizfragen zu einer Kategorie beantwortet werden müssen. Erstelle die GUI entsprechend der Abbildung:

**Programmablauf:**

Im obersten TextView wird die Kategorie angezeigt.

Darunter befindet sich ein Button der eine Frage anzeigt, gefolgt von vier schwarzen Buttons mit den verschiedenen Antworten.

Die grauen Buttons darunter sind disabled und zeigen den Status an.

Der Spieler klickt auf eine Antwort. Ist diese richtig wird der Button mit der Antwort grün eingefärbt, sonst rot. Alle Antwort-Buttons werden daraufhin disabled und der blaue Weiter-Button wird enabled.

Die 5 nebeneinander liegenden Buttons zeigen den Status an: grau bedeuten, dass die Frage noch nicht beantwortet wurde, grün, dass sie richtig beantwortet wurde und rot, dass sie falsch beantwortet wurde.

Durch Klicken des Weiter-Buttons wird die nächste Frage angezeigt, alle klickbaren Buttons werden wieder enabled bzw. disabled.

**Programmbeschreibung:****Datei `activity_main.xml`:**

Wähle ein Layout um die GUI entsprechend der Abbildung zu erstellen.

**Klasse `MainActivity.java`****`onDisplayQuestion()`**

Zeigt eine neue Frage und die dazugehörigen Antworten an und sorgt dafür, dass die richtigen Buttons enabled werden.

Setze einen **`OnClickListener`** auf den Weiter-Button, implementiert als innere, anonyme Klasse

**`onAnswerClick()`**

Implementiere den **`OnClickListener`** der vier schwarzen Antwort-Buttons als innere, Klasse.

Die Handler-Methode überprüft ob die Antwort richtig war und färbt den Antwort-Button und den jeweiligen Status-Button grün (richtig) bzw. rot (falsch) ein.

Die Antwort-Buttons werden disabled und der Weiter-Button wird enabled.

**Klasse `QuizQuestion.java`**

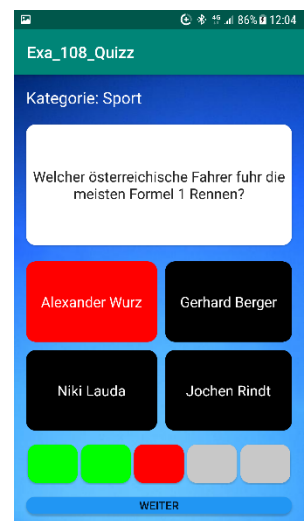
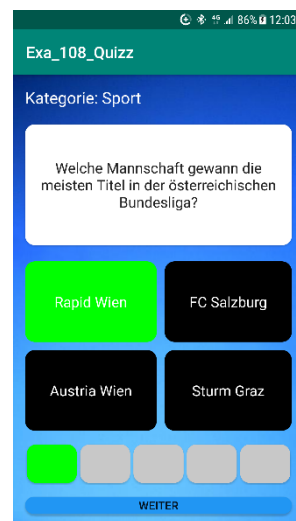
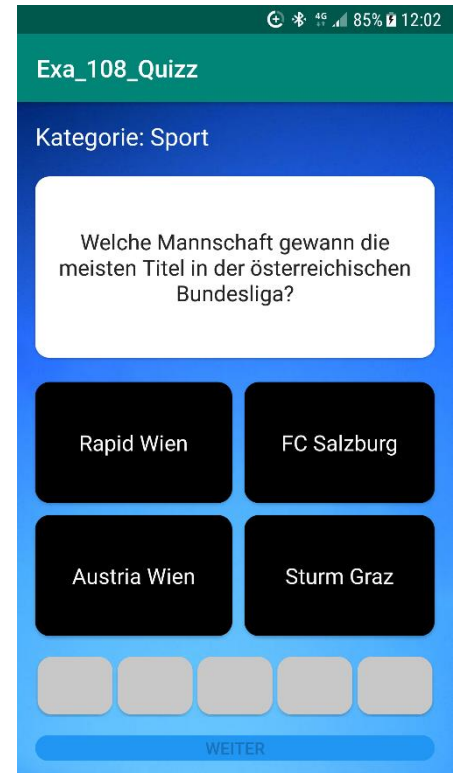
Entsprechend dem Klassendiagramm.

**Klasse `QuestionPool.java`**

Enthält alle Fragen in einer **`Map`**-Collection. Der Key für die **`Map`** ist der Name der Kategorie. Der Value der **`Map`** ist eine **`List`**-Collection, die zumindest fünf **`QuizQuestion`**-Objekte enthält.

**`List<QuizQuestion> getQuestionsByCategory(String category)`**

Übergeben wird der Name der Kategorie. Return-Wert ist die List mit allen Fragen (**`QuizQuestion`**-Objekten) dieser Kategorie.

**QuizQuestion**

- question : String
- answers : List<String>
- correctAnswer : int



**Hinweise:**

Um eine Bitmap als Hintergrund einzubinden, wird diese im Verzeichnis **res/drawable** gespeichert und über die Eigenschaft **background** als Ressource (@drawable/pic.jpg) dem Layout zugewiesen.

Um einen abgerundeten Button zu erhalten wird im Verzeichnis **res/drawable** die Datei **roundedbutton.xml** angelegt:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="#4CAF50" />
    <corners android:bottomRightRadius="12dp"
        android:bottomLeftRadius="12dp"
        android:topRightRadius="12dp"
        android:topLeftRadius="12dp"/>
</shape>
```

Um einen Button abgerundet anzuzeigen wird der Eigenschaft **background** der Wert der Ressource (@drawable/roundedbutton) zugewiesen.

Die Hintergrundfarbe für den Button wird über die Eigenschaft **backgroundTint** bestimmt.

Die Programmatische Änderung der Hintergrundfarbe für einen Button (z.B. **btStart**) erfolgt über folgende Methode:

```
ViewCompat.setBackgroundTintList(btStart,
    ColorStateList.valueOf(Color.GREEN));
```

Um über den Id-String einer View auf die dazugehörige Integer-Id zu kommen:

```
int btId = getResources().getIdentifier("btstart", "id", getPackageName());
```

*Kompetenzen:* Layout, Views, Event-Handling, Map-Collections

**Exa\_108\_NumberPuzzleGame**

Erstelle ein Spiel bei dem die, auf den Buttons abgebildeten Zahlen, aufsteigend sortiert werden. Die GUI ist entsprechend der Abbildung zu erstellen.

**Programmablauf:**

Die Zahlen von 1 bis 15 werden zufällig auf den Buttons verteilt.

Buttons mit geraden Zahlen bekommen einen roten Hintergrund, mit ungeraden Zahlen einen weißen Hintergrund und das leere Feld einen grauen Hintergrund.

Durch Wischen kann jeweils eine Zahl auf das leere Feld verschoben werden, wenn sie unmittelbar neben dem leeren Feld ist. Ziel des Spieles ist es die Zahlen von links nach rechts und von oben nach unten aufsteigend zu sortieren. Wenn das Spiel gewonnen wurde, werden alle Buttons disabled und die Meldung 'You won the game' über einen Toast angezeigt.

**Programmbeschreibung:****Datei `activity_main.xml`:**

Wähle ein Layout um die GUI entsprechend der Abbildung zu erstellen.

**Klasse `MainActivity.java`****`initButtons()`**

- Lege alle Buttons auf ein Feld das als Instanzvariable gespeichert wird. Initialisiere die Buttons mit zufälligen Zahlen von 1 bis 15 (alles verschiedene Zahlen) und setze die richtige Hintergrundfarbe: gerade Zahlen – roter Hintergrund, ungerade Zahlen – weißer Hintergrund, leered Feld – grauer Hintergrund.
- Setze einen `OnTouchListener` auf alle Buttons und delegiere das `onTouch`-Event an die `onFling()`-Methode der `GestureDetecture`-Klasse. Durch ein Wisch-Event nach links, rechts, oben oder unten kann ein Button auf das graue, freie Feld gezogen werden, wenn er unmittelbar benachbart zu diesem Feld ist. In der `onFling()`-Methode werden nur die Hintergründe der beiden Button ausgetauscht.
- Das Spiel ist beendet wenn die Zahlen in richtiger Reihenfolge angeordnet wurden. Es erfolgt eine Meldung mit einem Toast. Es kann kein weiterer Button mehr verschoben werden
- Implementiere einen `OnClickListener` für den Reset-Button um mit dem Spiel wieder von vorne beginnen zu können.

