1 Aktualisierung im Februar 2021

In der aktuellen Version 4.1.2 des Android Studio wird eine wichtige Bibliothek, mit deren Hilfe die Verbindung zwischen Benutzeroberfläche und Programmcode erstellt wird, nicht mehr zur Verfügung gestellt. Diese Bibliothek wird in fast allen Projekten des Buchs „Einstieg in Kotlin“ (ISBN 978-3-8362-6872-1) genutzt.

Als universelle Alternative wird im nachfolgenden Abschnitt 1.1 die Technik des ***View Binding*** anhand eines vollständigen Projekts ausführlich beschrieben. In den aktuellen Materialien zum Buch, die Ihnen auf dieser Webseite zur Verfügung stehen, wird diese Technik bereits in allen Projekten verwendet.

Dieser Abschnitt 1.1 ist eine aktualisierte Version des Buch-Abschnitts 2.7, »Ereignisse auslösen«.

Thomas Theis, im Februar 2021

1.1 Ereignisse auslösen

Button antippen

Die Benutzeroberfläche einer App stellt einem Benutzer Buttons und andere Bedienelemente zur Verfügung. Das Antippen eines Buttons oder das anderweitige Benutzen eines der Elemente ist ein Ereignis, das zu einem Programmablauf führt.

View Binding

In diesem Abschnitt wird das Projekt mit dem Namen ***Ereignisse*** vorgestellt. Darin wird mithilfe der Technik des ***View Binding*** eine Verbindung zwischen Benutzeroberfläche und Programmablauf erstellt.

1.1.1 Die fertige App

Zwei Buttons

Erstellen Sie wie gewohnt ein neues Projekt mit dem Namen ***Ereignisse***. Mithilfe dieses Projekts wird die gleichnamige App erstellt. Ihre Benutzeroberfläche beinhaltet zwei Buttons mit den eindeutigen IDs `buDeutsch` beziehungsweise `buEnglisch` sowie die TextView `tvAusgabe` mit dem Text `Ausgabe` in einem vertikalen LinearLayout (siehe Abbildung 1.1).

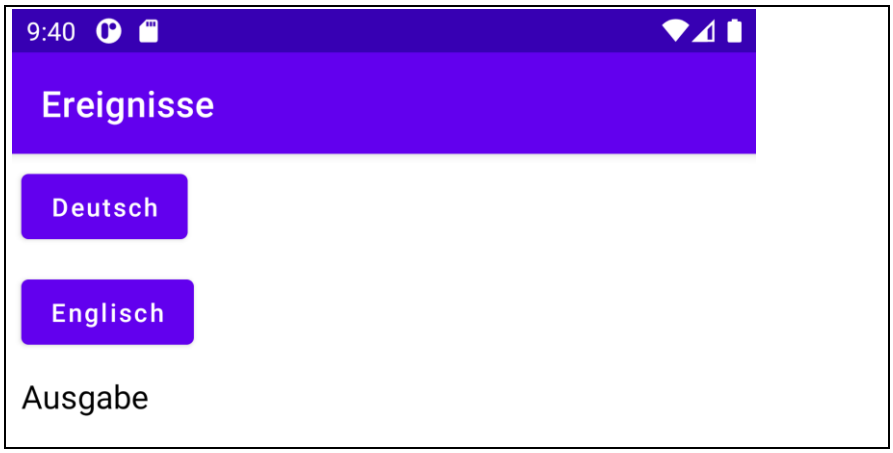

Abbildung 1.1: App Ereignisse

Reaktion auf Ereignis

Nach der Betätigung eines der Buttons wird entweder der Text `Guten Morgen` oder der Text `Good morning` in der TextView angezeigt.

1.1.2 Die Ressourcen für die Benutzeroberfläche

Es folgt der XML-Code für die verschiedenen Ressourcen, die zur Erstellung der Benutzeroberfläche benötigt werden. Die Erstellung von Ressourcen wird im Buch-Abschnitt 2.5, »Layout und Ressourcen« beschrieben. Zunächst die Farb-Ressourcen:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    ...
    <color name="white">#FFFFFFFF</color>
    <color name="textFarbe">#000000</color>
</resources>
```

Listing 1.1: Projekt Ereignisse, Datei colors.xml

Es folgen die Ressourcen für die Dimensionen:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textGroesse">18sp</dimen>
    <dimen name="abstand">5dp</dimen>
</resources>
```

Listing 1.2: Projekt Ereignisse, Datei dimens.xml

Zu guter Letzt folgen die String-Ressourcen:

```
<resources>
    <string name="app_name">Ereignisse</string>
```

```
    <string name="buDeutschString">Deutsch</string>
    <string name="buEnglischString">Englisch</string>
    <string name="tvAusgabeString">Ausgabe</string>
</resources>
```

Listing 1.3: Projekt Ereignisse, Datei strings.xml

1.1.3 Das Layout der Benutzeroberfläche

In diesem Abschnitt folgt der XML-Code zur Erstellung der Benutzeroberfläche, der in der Datei **activity_main.xml** steht, siehe auch Buch-Abschnitt 2.2.2.

Es werden die Ressourcen aus Abschnitt 1.1.2 genutzt:

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

    <Button
        android:id="@+id/buDeutsch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/abstand"
        android:text="@string/buDeutschString"
        android:textAllCaps="false" />

    <Button
        android:id="@+id/buEnglisch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/abstand"
        android:text="@string/buEnglischString"
        android:textAllCaps="false" />

    <TextView
        android:id="@+id/tvAusgabe"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_margin="@dimen/abstand"
        android:text="@string/tvAusgabeString"
        android:textColor="@color/textFarbe"
        android:textSize="@dimen/textGroesse" />
</LinearLayout>
```

Listing 1.4: Projekt Ereignisse, Datei activity_main.xml

Anhand der jeweiligen Abbildung der fertigen App lässt sich leicht erkennen, wie die Benutzeroberfläche erstellt wird. Daher wird der jeweilige XML-Code in den nachfolgenden Projekten nur dargestellt, falls darin neue Elemente vorkommen.

1.1.4 Klassen und Objekte

Objektorientiert

Zum besseren Verständnis des Programmcodes versorge ich Sie zunächst mit ein wenig Theorie. In Kotlin arbeiten wir unter anderem nach dem Prinzip der ***Objektorientierten Programmierung*** (kurz: ***OOP***). Einige wichtige Begriffe in der OOP sind ***Klasse***, ***Objekt***, ***Eigenschaft***, ***Methode*** und ***Vererbung***.

Klasse, Datentyp, Typ

Es werden Softwareobjekte erzeugt, die realen Objekten nachgebildet werden. Als Muster zur Erzeugung gleichartiger Objekte werden zunächst Klassen entworfen. In einer Klasse werden Eigenschaften und Methoden definiert. Häufig wird statt des Begriffs ***Klasse*** auch der Begriff ***Datentyp*** oder kurz ***Typ*** verwendet.

Objekte

Nach der Definition einer Klasse können beliebig viele Objekte dieser Klasse erzeugt werden. Die einzelnen Objekte besitzen individuelle Werte für die verschiedenen Eigenschaften. Ihr Verhalten richtet sich nach den definierten Methoden. Mehr zum Thema ***Klassen*** gibt es in Kapitel 10, »Eigene Klassen«.

Vererbung

Das Prinzip der Vererbung ermöglicht den Aufbau von eigenen Klassenbibliotheken, die eine aufeinander abgestimmte Hierarchie von Klassen beinhalten. Zum Thema ***Vererbung*** finden Sie mehr im gleichnamigen Abschnitt 10.6. Für die Programmierung in Kotlin stehen zahlreiche vordefinierte Klassen in Klassenbibliotheken zur Verfügung. Es können Objekte dieser Klassen erzeugt werden. Zudem können eigene Klassen erzeugt werden, die von einer dieser Klassen erben.

Singleton

In Kotlin wird häufig mit ***Singleton***-Objekten (kurz: Singletons) gearbeitet. Ihr Entwurf ähnelt dem einer Klasse. Allerdings gibt es anschließend nur ein einziges Objekt. Singleton-Objekte sind einzigartig und ermöglichen einen eindeutigen Zugriff.

1.1.5 Eine neue Klasse für das View Binding

In der Einleitung zu dieser »Aktualisierung im Februar 2021« wird die Technik des ***View Binding*** erwähnt, mit deren Hilfe die Verbindung zwischen Benutzeroberfläche und Programmcode erstellt werden soll. Zur Anwendung dieser Technik innerhalb eines Projekts muss zunächst eine Änderung der Konfiguration des Projekts vorgenommen werden.

Wechseln Sie im Projektfenster links oben auf die Ansicht **PROJECT**. Klappen Sie die einzelnen Elemente der Hierarchie auf, bis die Datei **build.gradle** im Verzeichnis **app** zu sehen ist, siehe Abbildung 1.2. Achten Sie darauf, die richtige Datei auszuwählen: Es gibt mehrere Dateien dieses Namens in unterschiedlichen Verzeichnissen.

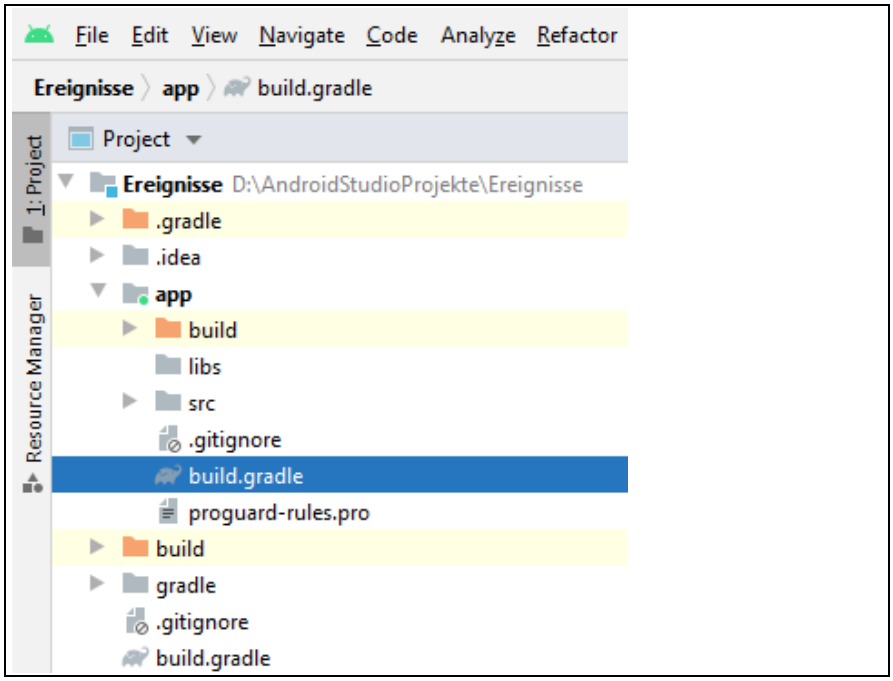

Abbildung 1.2: Datei app/build.gradle in der Hierarchie

Nach einem Doppelklick auf die Datei **app/build.gradle** erscheint diese rechts im Codefenster. Fügen Sie im Codeblock `android` den Codeblock `buildFeatures` unterhalb des Codeblocks `kotlinOptions` hinzu, wie es in Listing 1.5 zu sehen ist.

```
plugins {
    ...
}

android {
    ...
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures {
        viewBinding true
    }
}

dependencies {
    ...
}
```

Listing 1.5: Datei app/build.gradle

Nach dieser Änderung wird am oberen Rand des Codefensters eine Meldung eingeblendet. Sie werden aufgefordert, die Teile des Projekts neu miteinander zu verbinden, indem sie den Hyperlink **SYNC NOW** rechts neben der Meldung betätigen. Führen Sie dies durch.

Anschließend steht Ihnen in diesem Projekt die automatisch erzeugte Klasse `ActivityMainBinding` zur Verfügung, die Ihnen mithilfe der Technik des ***View Binding*** die Elemente der Benutzeroberfläche aus der Datei **activity_main.xml** bereithält. Im nächsten Abschnitt wird diese Klasse genutzt.

1.1.6 Der Coderahmen

Datei MainActivity.kt

Lassen Sie sich die Datei **MainActivity.kt** anzeigen, siehe auch Buch-Abschnitt 2.2.1. Sie beinhaltet zunächst denjenigen Kotlin-Code, der bei der Erstellung eines Projekts automatisch erzeugt wird.

Passen Sie diesen Code an, wie es in Listing 1.6 angegeben ist. Die drei Punkte im Listing stehen für weiteren Code, der in Abschnitt 1.1.7 eingefügt wird.

```
package com.example.ereignisse

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.example.ereignisse.databinding.ActivityMainBinding
```

```
class MainActivity : AppCompatActivity() {
    private lateinit var B: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        B = ActivityMainBinding.inflate(layoutInflater)
        setContentView(B.root)
        ...
    }
}
```

Listing 1.6: Projekt Ereignisse, Datei MainActivity.kt

Der Programmcode und die zugehörigen Erläuterungen erscheinen zunächst sehr umfangreich. Sie müssen jetzt auch noch nicht alles verstehen. Es wird ein Grundaufbau zum Erlernen von Kotlin mithilfe von Apps erstellt, der auf diese Art und Weise in sehr vielen Projekten genutzt wird. Mit wachsendem Verständnis für die Programmierung werden Ihnen die einzelnen Elemente vertraut.

package

Nach dem Schlüsselwort `package` folgt der eindeutige Name des Pakets dieses Projekts, wie er bei der Erstellung des Projekts angegeben wurde.

import

Weitere vordefinierte Pakete mit Klassenbibliotheken oder einzelne Klassen, die in diesem Projekt benötigt werden, lassen sich mithilfe des Schlüsselworts `import` verfügbar machen. Unter anderem wird die Klasse `ActivityMainBinding` eingebunden, die Sie in Abschnitt 1.1.5 durch die Änderung der Konfiguration des Projekts erzeugt haben.

Klasse MainActivity

In der Klasse `MainActivity` werden die Eigenschaften und das Verhalten der Hauptaktivität der App beschrieben. Die Definition einer Klasse wird mit dem Schlüsselwort `class` und dem Namen der Klasse eingeleitet. Ihr Code folgt in einem Block, der mithilfe von geschweiften Klammern gebildet wird. Nachfolgend sehen Sie den vereinfachten Aufbau einer Klasse:

```
class [Name der Klasse] {
[Code der Klasse]
}
```

Klasse AppCompatActivity

Die Klasse `MainActivity` erbt in unseren Projekten von der Basisklasse `AppCompatActivity`. Auf diese Weise wird für die Abwärtskompatibilität des Programmcodes und die Unterstützung für ältere Geräte gesorgt. Nachfolgend sehen Sie den vereinfachten Aufbau einer Klasse, die von einer Basisklasse erbt:

```
class [Name der Klasse]:[Name der Basisklasse] {
[Code der Klasse]
}
```

Objekt erzeugen

Mit der Anweisung `private lateinit var B: ActivityMainBinding` wird ein Objekt der Klasse `ActivityMainBinding` erstellt, mit dessen Hilfe die Elemente der Benutzeroberfläche in der Datei **activity_main.xml** erreicht werden können. Der Anweisungsteil `var B` sorgt dafür, dass die Variable `B` einen Verweis auf dieses Objekt darstellt.

lateinit

Mithilfe des Schlüsselworts `private` wird dieses Objekt innerhalb der Klasse `MainActivity` geschützt und ist nur innerhalb dieser Klasse erreichbar. Normalerweise müssen Variablen zu Beginn initialisiert werden; es muss ihnen also unmittelbar ein Wert zugewiesen werden. Das Schlüsselwort `lateinit` (kurz für: ***late initialization***, deutsch: späte Initialisierung) erlaubt es, den Wert erst später zuzuweisen.

onCreate()

Der Code, der nach dem Start der App ausgeführt wird, befindet sich in der Methode `onCreate()`. Eine Methode ist eine Funktion, die für ein Objekt ausgeführt wird. Die Definition einer Funktion (oder einer Methode) wird mit dem Schlüsselwort `fun` und dem Namen der Funktion eingeleitet. Ihr Code folgt in einem Block aus geschweiften Klammern. Im Buch-Kapitel 4 (Funktionen) folgen ausführliche Erläuterungen zu diesem Thema.

Nachfolgend sehen Sie den vereinfachten Aufbau einer Funktion:

```
fun [Name der Funktion]:[Typ des Rückgabewerts der Funktion]
            ([Liste der Parameter der Funktion]) {
    [Code der Funktion]
}
```

super

Die Methode `onCreate()` wird von der Basisklasse `AppCompatActivity` geerbt. In der Klasse `MainActivity` wird sie mithilfe des Schlüsselworts `override` überschrieben, das heißt mit eigenem Code gefüllt. Mithilfe des Schlüsselworts `super` wird zusätzlich die geerbte Methode der Basisklasse direkt aufgerufen.

inflate()

Die Methode `inflate()` sorgt dafür dass das Layout ***aufgepumpt*** wird. Anschließend steht die vollständige Hierarchie dieses Layouts aus der Datei **activity_main.xml** über den Verweis `B` zur Verfügung.

setContentView()

Der Aufruf der Methode `setContentView()` greift auf das Wurzelelement dieser Hierarchie zu. Damit wird erreicht, dass das betreffende Layout in dieser Activity dargestellt wird.

1.1.7 Die Erweiterung des Coderahmens

Eigener Code

Damit das Antippen der beiden Buttons zu Ereignissen führt, wird der Coderahmen erweitert. Anstelle der drei Punkte in Listing 1.6 wird der nachfolgende Code eingefügt:

```
/* Code nach Antippen des
   Buttons buDeutsch */
B.buDeutsch.setOnClickListener {
    B.tvAusgabe.text = "Guten Morgen"
}

// Button buEnglisch
B.buEnglisch.setOnClickListener {
    B.tvAusgabe.text = "Good morning"
}
```

Listing 1.7: Datei MainActivity.kt, Coderahmen, Erweiterung

Zugriff auf Layoutelemente

Über den Verweis B wird auf die Elemente der Benutzeroberfläche zugegriffen. Die Eigenschaften `buDeutsch`, `buEnglisch` und `tvAusgabe` stehen für die gleichnamigen Elemente aus der XML-Datei.

Listener

Bei einem ***Listener*** (dt.: ***Horcher***) handelt es sich um ein Objekt, das ein Ereignis bemerkt und mithilfe seines Codes darauf reagieren kann. Der Aufruf der Methode `setOnClickListener()` legt fest, welches Listener-Objekt auf das Ereignis ***Klick auf Button*** beziehungsweise ***Antippen des Buttons*** reagiert.

Anonyme Funktion

Der Code eines Listener-Objekts kann auf unterschiedliche Arten notiert werden. In vielen Fällen genügt die hier vorliegende Kurzschreibweise, und zwar der Aufruf einer sogenannten ***anonymen Funktion***: Innerhalb eines Blocks aus geschweiften Klammern stehen diejenigen Anweisungen, die nach dem Auslösen des Ereignisses ausgeführt werden. Mehr zu anonymen Funktionen finden Sie im gleichnamigen Buch-Abschnitt 4.7.

Reaktion auf Ereignis

Nach dem Aufruf der Methode `setOnClickListener()` für den Button `buDeutsch` wird dafür gesorgt, dass nach einem Antippen des Buttons die Eigenschaft `text` der TextView `tvAusgabe` geändert wird. Das Entsprechende wird für den zweiten Button durchgeführt.

Listener-Varianten

Nach dem Kennenlernen weiterer Elemente der Sprache Kotlin zeige ich im Projekt ***ListenerVarianten*** im Buch-Abschnitt 5.2, »Listener«, zusätzliche Möglichkeiten für Listener-Objekte. Zudem erläutere ich dort, wie die verschiedenen Varianten zusammenhängen und wie Sie

schrittweise zu der hier genutzten Variante mit der anonymen Funktion gelangen.

1.1.8 Anweisungen und Kommentare

Mehrzeilig

Jede Anweisung steht in einer eigenen Textzeile. Ist eine Anweisung zu lang, zum Beispiel für die Darstellung im Buch, kann sie auf mehrere Zeilen verteilt werden. Allerdings ist nicht jede Stelle zur Auftrennung der Anweisung geeignet. Eine falsche Auftrennung wird unmittelbar angezeigt, indem der fehlerhafte Code in roter Farbe dargestellt wird.

Semikolon

Ein Semikolon am Ende einer Anweisung ist nur erforderlich, wenn Sie mehrere Anweisungen in einer Zeile notieren möchten.

Kommentare

Programme sollten in ausreichender Form kommentiert werden. Diese Kommentare werden nicht ausgeführt und dienen nur der Erläuterung des betreffenden Programmteils. Das erweist sich als sehr nützlich bei einer späteren Änderung des Codes durch dieselbe oder eine andere Person.

/* … */
//

Längere Kommentare können sich über mehrere Zeilen erstrecken und werden zwischen den Zeichenfolgen `/*` und `*/` eingebettet. Kürzere Kommentare werden nach der Zeichenfolge `//` notiert und gehen nur bis zum Ende der jeweiligen Zeile.

1.1.9 Weiterentwicklung und Neustart

Schrittweise Entwicklung

Geben Sie den Code nach und nach in Ihr Projekt ein. Starten Sie das Projekt zum Testen bereits jeweils nach den einzelnen Programmteilen, etwa nach dem Hinzufügen der Elemente für den ersten Button, der Elemente für den zweiten Button und so weiter. Diese schrittweise Vorgehensweise erleichtert Ihnen die Fehlersuche, sollte die Ausgabe nicht den Erwartungen entsprechen.

Apply Changes

Nach dem ersten Start einer App in einem Gerät müssen Sie sie für einen erneuten Start nicht zuvor beenden. Stattdessen rufen Sie den Menüpunkt **Run • Apply Changes and Restart Activity** auf. Sie finden das zugehörige Symbol mit dem Pfeil und dem kleinen Buchstaben **A** auch in der Symbolleiste neben dem Symbol für **Run app**. Nach einer Änderung des Projekts werden nur die geänderten Teile eines Projekts aktualisiert. Der Neustart gelingt in den meisten Fällen viel schneller.

Ein Neustart ist auch nützlich, falls Sie beim Testen Ihrer App einige Elemente bedient und damit das Aussehen der App geändert haben. Anschließend sehen Sie sie wieder im Startzustand.

1.1.10 Android- und Kotlin-Hilfe

Startseite

Sie finden allgemein Hilfe zur Entwicklung mit Kotlin im Android Studio über die Adresse *https://developer.android.com*. Auf der Startseite finden Sie das übliche Suchfeld, um sich innerhalb der Hilfe umzuschauen.

Suchmaschine

Benötigen Sie direkt eine Kotlin-bezogene Hilfe zu einem bestimmten Element, erweist sich auch die nachfolgende Eingabe in einer Suchmaschine als sehr schnell und erfolgreich:

```
developer kotlin <Suchbegriff>
```

Ein Beispiel: Sie benötigen Hilfe zum Begriff ***TextView*** und geben ein:

```
developer kotlin textview
```

Anschließend wird Ihnen meist als erstes Suchergebnis eine Seite angezeigt, die Erläuterungen zur Klasse `TextView` bietet: *https://developer.android.com/reference/kotlin/android/widget/TextView*

1.1.11 Hilfen des Editors

Der Editor des Android Studio bietet Hilfestellungen zum Entwickeln, zum Beispiel gezeigt in Abbildung 1.3. Das beinhaltet auch Anmerkungen an einigen Stellen meines Kotlin-Codes, siehe dazu Buch-Abschnitt 1.2, »Aufbau dieses Buchs«.

Deutsche Texte

Bei eigenen Bezeichnungen nutze ich häufig selbsterklärende deutsche Begriffe. Zudem lasse ich viele Texte unmittelbar ausgeben, ohne den ***Umweg*** über eine String-Ressource. Eine andere Vorgehensweise würde die Übersetzung des Programms in andere Sprachen erleichtern. Das ist aber nicht unser vorrangiges Ziel.

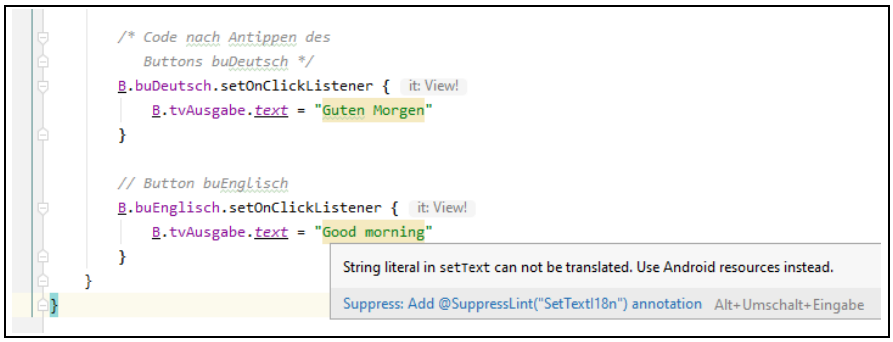

Abbildung 1.3: Hilfestellungen

Parameter it

Innerhalb des Codes der anonymen Funktion für den Listener steht der Parameter `it` zur Verfügung. Er stellt einen Verweis auf die aktuelle View dar, also auf den Button `buDeutsch` beziehungsweise den Button

`buEnglisch`. Zur Hilfestellung blendet der Editor alle Parameter jeweils mit seinem Namen und seinem Datentyp ein. Auch hier verweise ich zum besseren Verständnis auf den Buch-Abschnitt 5.2, »Listener«.