

To-Do-List

Appdokumentation

Thema des Projektes: Entwicklung einer Java-Applikation in Android mit der Möglichkeit, Daten zu speichern

Autor: Daniel Ermilov

Inhaltsverzeichnis

[1 Einleitung 4](#_Toc180055268)

[1.1 Ziel und Zweck 4](#_Toc180055269)

[1.2 Arbeitsumgebung 4](#_Toc180055270)

[2 Überblick 4](#_Toc180055271)

[2.1 Systemarchitektur 4](#_Toc180055272)

[2.2 Programmiersprachen 4](#_Toc180055273)

[2.3 Aufbau 4](#_Toc180055274)

[2.4 Notwendige Einstellungen 5](#_Toc180055275)

[2.5 Ablaufdiagramm 7](#_Toc180055276)

[2.6 Schnittstellen 8](#_Toc180055277)

[3 Implementieren von Code 8](#_Toc180055278)

[3.1 Implementieren der Klasse Aufgabe 8](#_Toc180055279)

[3.1.1 Attribute 8](#_Toc180055280)

[3.1.2 Konstruktor 9](#_Toc180055281)

[3.1.3 Getter/Setter 9](#_Toc180055282)

[3.2 Implementieren von Klasse AppDatenbank 9](#_Toc180055283)

[3.2.1 Methode aufgabenDao 10](#_Toc180055284)

[3.3 Implementieren vom Interface AufgabenDao 10](#_Toc180055285)

[3.3.1 SQL-Methoden 11](#_Toc180055286)

[3.4 Implementieren der Klasse OffeneAufgabenAdapter 11](#_Toc180055287)

[3.4.1 Attribute 11](#_Toc180055288)

[3.4.2 Konstruktor 12](#_Toc180055289)

[3.4.3 Setter 12](#_Toc180055290)

[3.4.4 Methode onCreateViewHolder 12](#_Toc180055291)

[3.4.5 Methode getItemCount 13](#_Toc180055292)

[3.4.6 Statische Klasse OffeneAufgabenHolder 13](#_Toc180055293)

[3.4.7 Methode onBindViewHolder 13](#_Toc180055294)

[3.5 Implementieren der Klasse ErledigteAufgabenAdapter 15](#_Toc180055295)

[3.5.1 Attribute 15](#_Toc180055296)

[3.5.2 Konstruktor 15](#_Toc180055297)

[3.5.3 Methode updateListen 16](#_Toc180055298)

[3.5.4 Methode onCreateViewHolder 16](#_Toc180055299)

[3.5.5 Methode getItemCount 16](#_Toc180055300)

[3.5.6 Statische Klasse ErledigteAufgabenHolder 16](#_Toc180055301)

[3.5.7 Methode onBindViewHolder 17](#_Toc180055302)

[3.6 Implementieren der Klasse Alarm 19](#_Toc180055303)

[3.6.1 Attribute 19](#_Toc180055304)

[3.6.2 Konstruktor 19](#_Toc180055305)

[3.6.3 Setter 20](#_Toc180055306)

[3.6.4 Methode updateListen 20](#_Toc180055307)

[3.6.5 Methode alertAufgabeDetails 20](#_Toc180055308)

[3.6.6 Methode alertBestaetigeLoeschvorgang 21](#_Toc180055309)

[3.6.7 Methode alertNeueAufgabeErstellen 22](#_Toc180055310)

[3.6.8 Methode alertEmptyTitel 23](#_Toc180055311)

[3.6.9 Methode updateUI 23](#_Toc180055312)

[3.7 Implementieren der Klasse MainActivity 24](#_Toc180055313)

[3.7.1 Attribute 24](#_Toc180055314)

[3.7.2 Methode onCreate 25](#_Toc180055315)

[3.7.3 Methode ladeAufgabenUndUpdateUI 26](#_Toc180055316)

[3.7.4 Methode leseDatenbankUndAktualisierListen 27](#_Toc180055317)

[3.7.5 Methode aktualisiereViewSichtbarkeit 27](#_Toc180055318)

[3.8 Ressourcen 28](#_Toc180055319)

[3.8.1 Drawable 28](#_Toc180055320)

[3.8.2 Layouts 28](#_Toc180055321)

[3.8.3 Values 28](#_Toc180055322)

[4 Nützliche Links 29](#_Toc180055323)

[5 Glossar 29](#_Toc180055324)

# Einleitung

## Ziel und Zweck

Diese Anwendung wurde entwickelt, um die Benutzer bei der Organisation und Verfolgung ihrer Aufgaben zu unterstützen.

## Arbeitsumgebung

Die genutzte Entwicklungsumgebung ist Android Studio, eine von Google entwickelte Plattform, die das Erstellen und Designen von Apps vereinfacht. Android Studio beinhaltet zahlreiche Emulatoren, welche die Testmöglichkeit auf diversen Geräten ermöglichen. In diesem Rahmen wurde ein neuer Workspace sowie einige Klassen erstellt.

# Überblick

## Systemarchitektur

Die To-Do App nutzt die in Android integrierte Root-Datenbank (SQLite), um Aufgaben lokal auf dem Gerät des Benutzers zu speichern. Diese Datenbank ermöglicht es der App, die Daten dauerhaft zu speichern, so dass die Aufgaben auch nach dem Schließen oder Neustarten der App verfügbar bleiben.

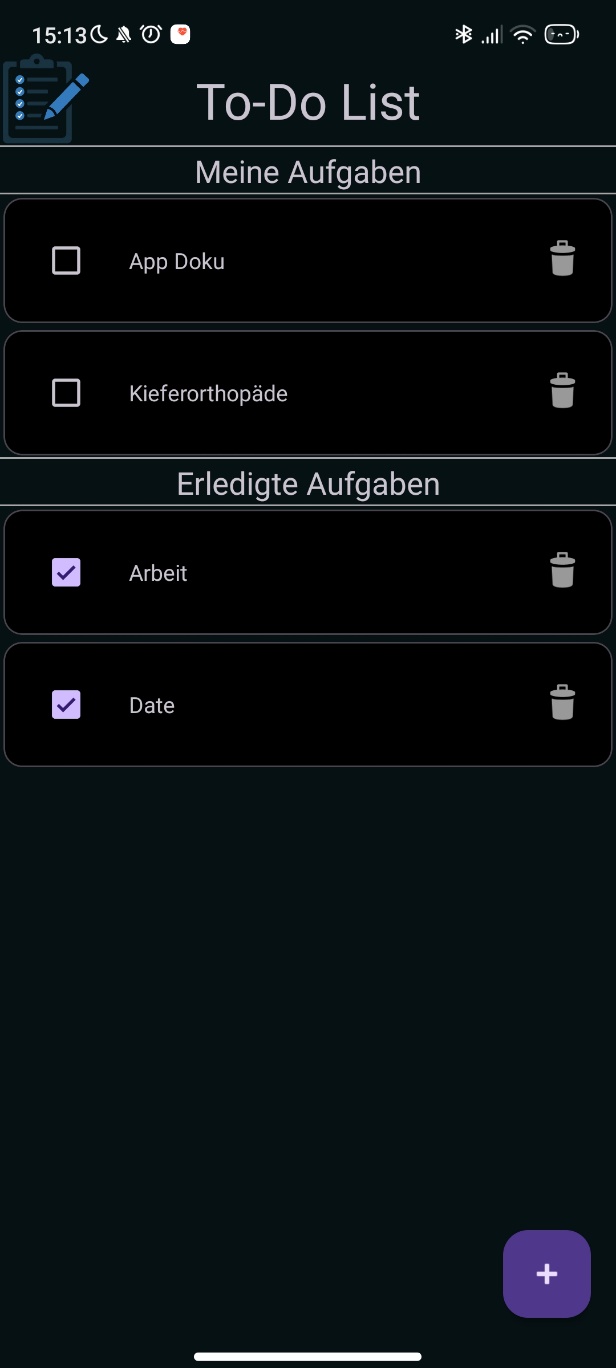
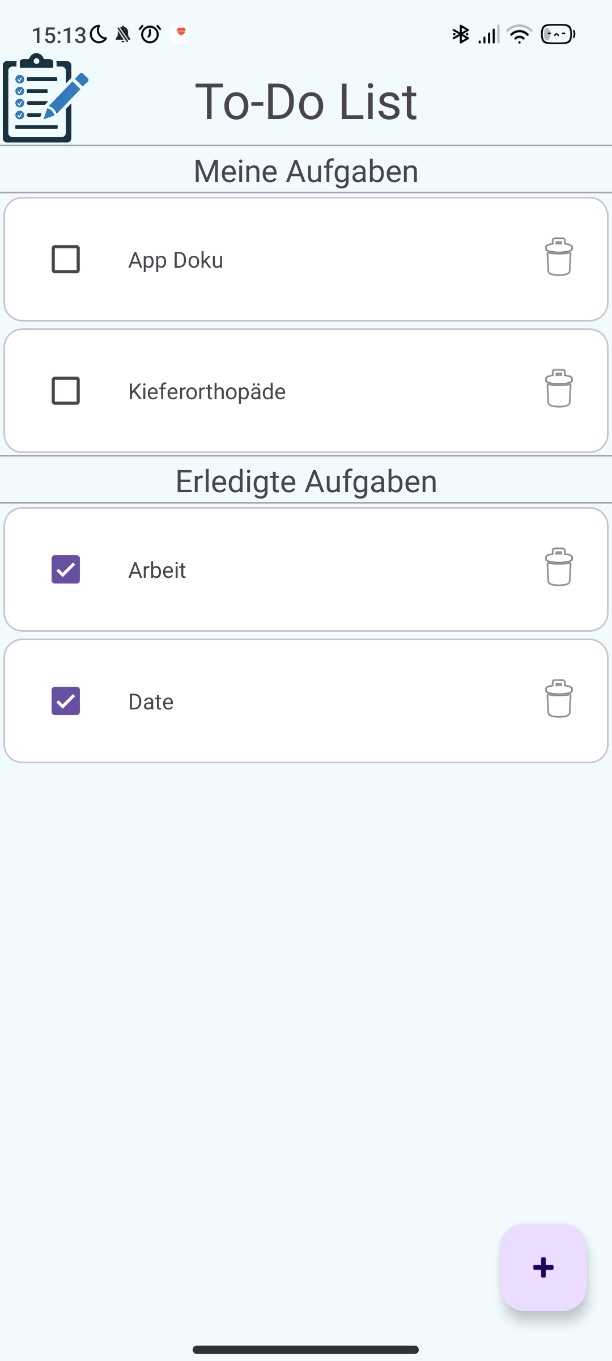
## Programmiersprachen

Die To-Do App wurde in Java programmiert und verwendet XML für das Design der Benutzeroberfläche. SQL wird verwendet, um mit der Datenbank zu interagieren und Daten abzufragen oder zu ändern.

## Aufbau

Diese Anwendung verfügt über 2 Themes. Diese Themes wurden erstellt, um die App zu gestalten. Ein Theme wird für den Lightmode und das andere für den Darkmode verwendet.

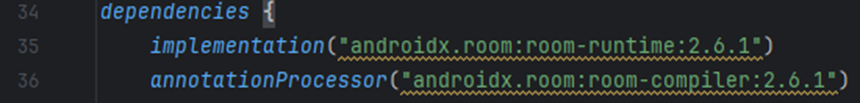
Android wählt automatisch das Theme aus, in dessen Modus sich das System befindet.



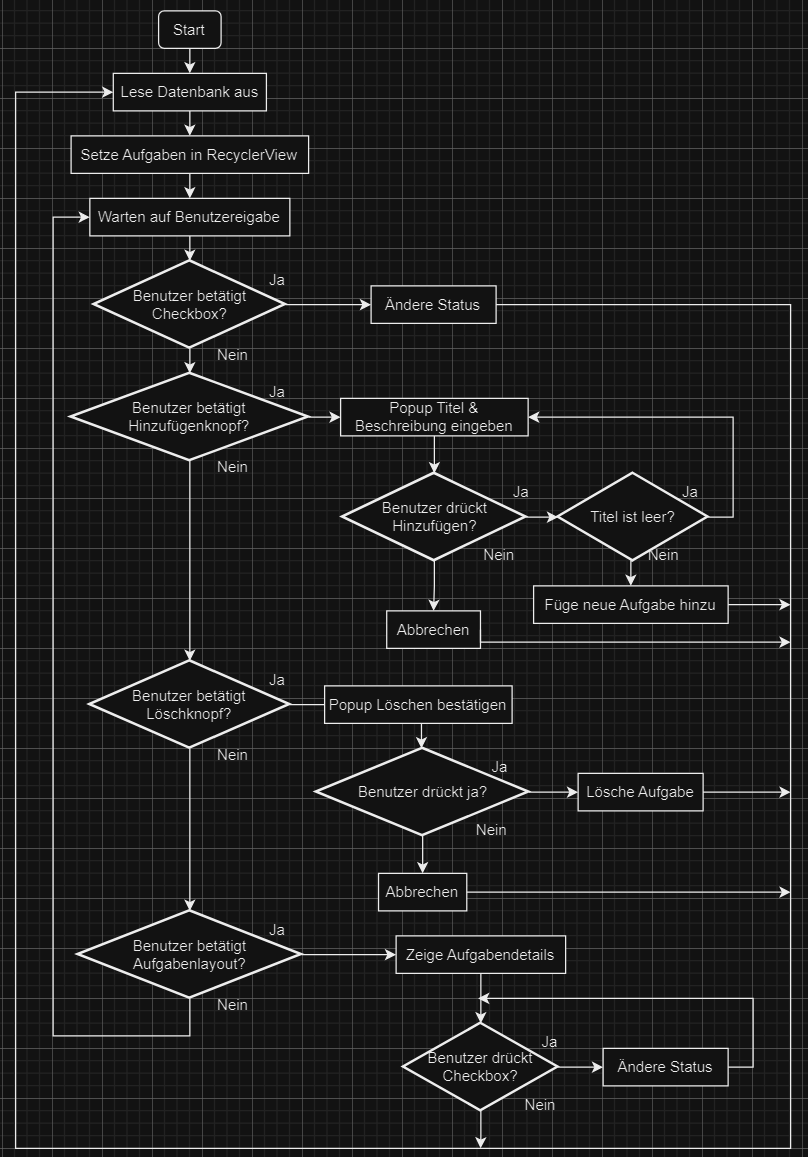
Darkmode Lightmode

## Notwendige Einstellungen

Bevor eine Room-Datenbank eingefügt werden kann, müssen einige Implementierungen vorgenommen werden. Diese müssen in build.gradle (Module) unter dependencies eingetragen werden. Room ist eine Android Bibliothek, die eine zusätzliche Struktur über SQLite zur Verfügung stellt und somit die Arbeit mit Datenbanken erleichtert.



## Ablaufdiagramm



## Schnittstellen

Technische Schnittstellen

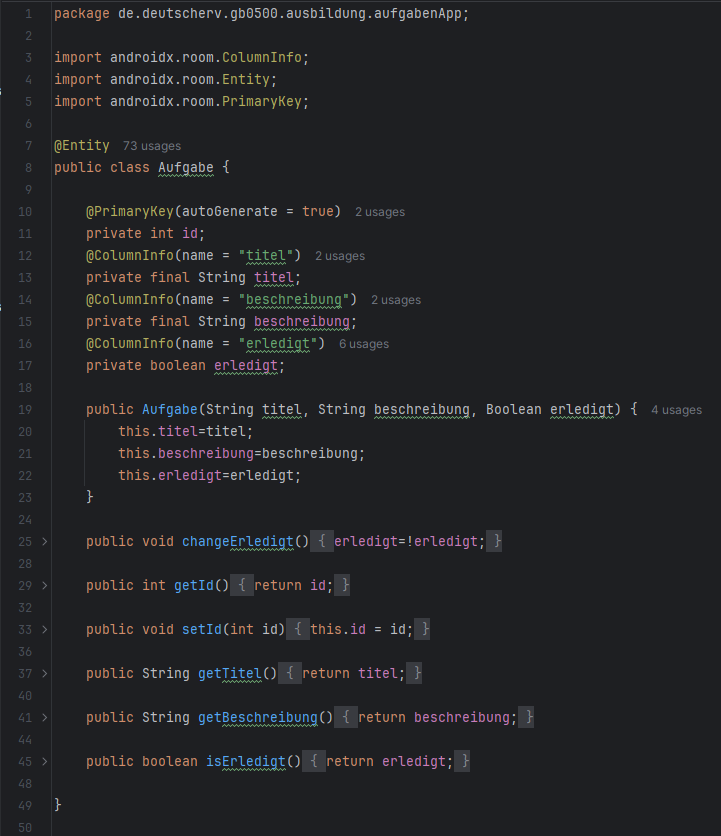
* Notebook mit Windows 11
* Android Studio als Entwicklungsumgebung
* SDK 24 (Android 7.0 / Nougat)
* Microsoft 365 zum Erstellen von Dokumentation
* [www.draw.io](http://www.draw.io) zum Erstellen eines Ablaufdiagramms

# Implementieren von Code

Um die Eigenschaften unserer Objekte in Java zu spezifizieren, werden Klassen erstellt. Klassen ermöglichen eine logische Strukturierung und Organisation des Codes. Sie fördern die Wiederverwendbarkeit des Codes. Dadurch wird Redundanz vermieden und der Code wird wartbarer. Die Verwendung von Klassen erleichtert das Testen des Codes. Einzelne Klassen können isoliert getestet werden, um sicherzustellen, dass sie wie erwartet funktionieren.

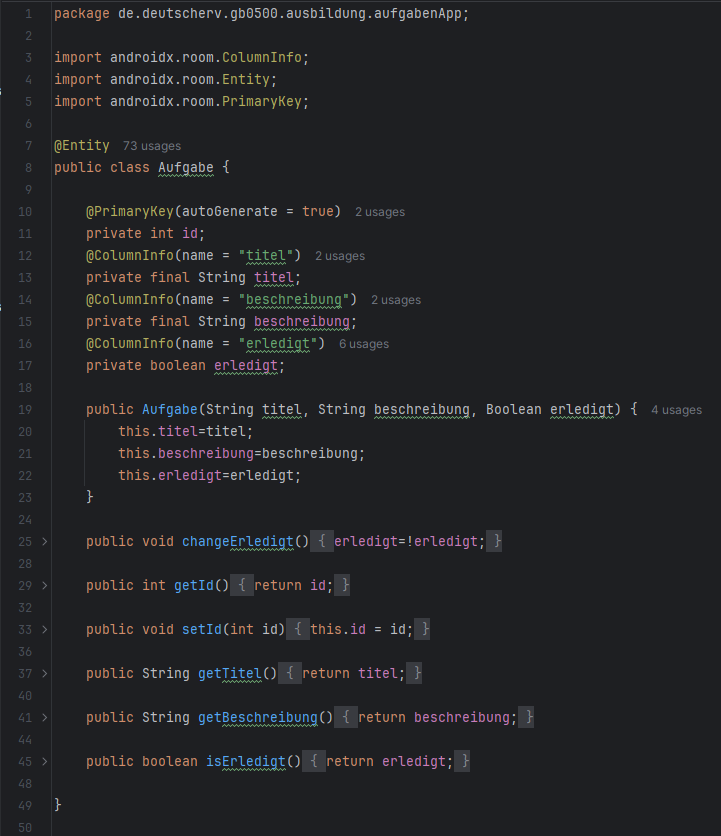
## Implementieren der Klasse Aufgabe

Die Room-Datenbank arbeitet mit Entitys. Damit die Datenbank versteht, welche Klassen/Objekte in der Datenbank verwendet werden, schreibt man @Entity über die Klassendeklaration. Dies ist später bei der Erstellung der Datenbank wichtig.



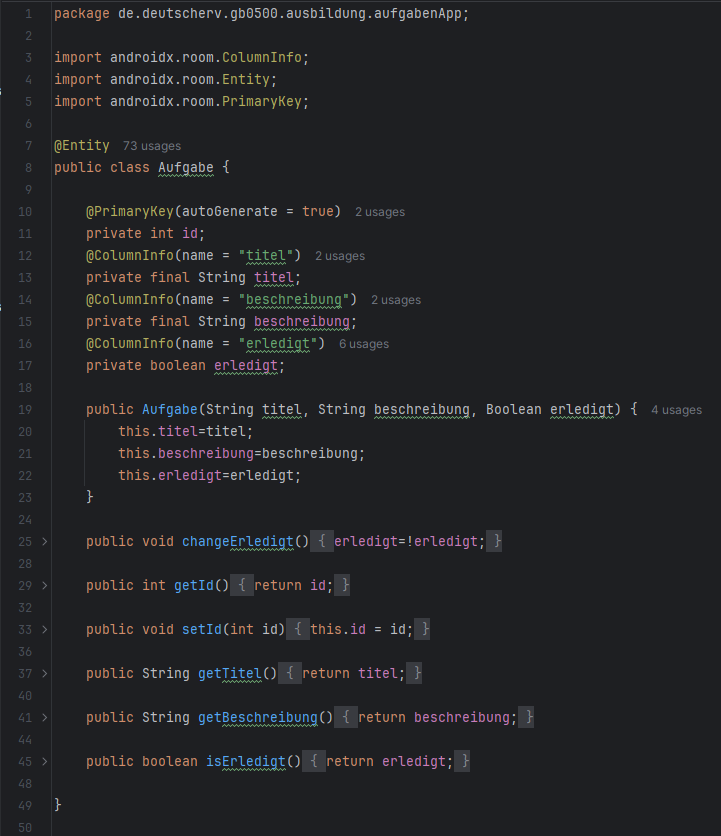
### Attribute

In dieser Klasse wurden die Attribute ID, Titel, Beschreibung und erledigt hinzugefügt. Die ID ist nur für die Datenbank wichtig, um Objekte mit einem Wert zu übergeben, der nicht wiederholt wird. Dazu muss über der Attributdeklaration @PrimaryKey(autoGenerate = true) eingetragen werden. Das autoGenerate = true sorgt dafür, dass die Datenbank beim Anlegen eine ID für dieses Objekt generiert und verwendet, die nicht doppelt vorkommt. Für die anderen Attribute wurde @CollumnInfo(name = [Attributname]) eingetragen. Dies sorgt dafür, dass die Datenbank für das Objekt eine Spalte mit dem jeweiligen Attributnamen erzeugt.



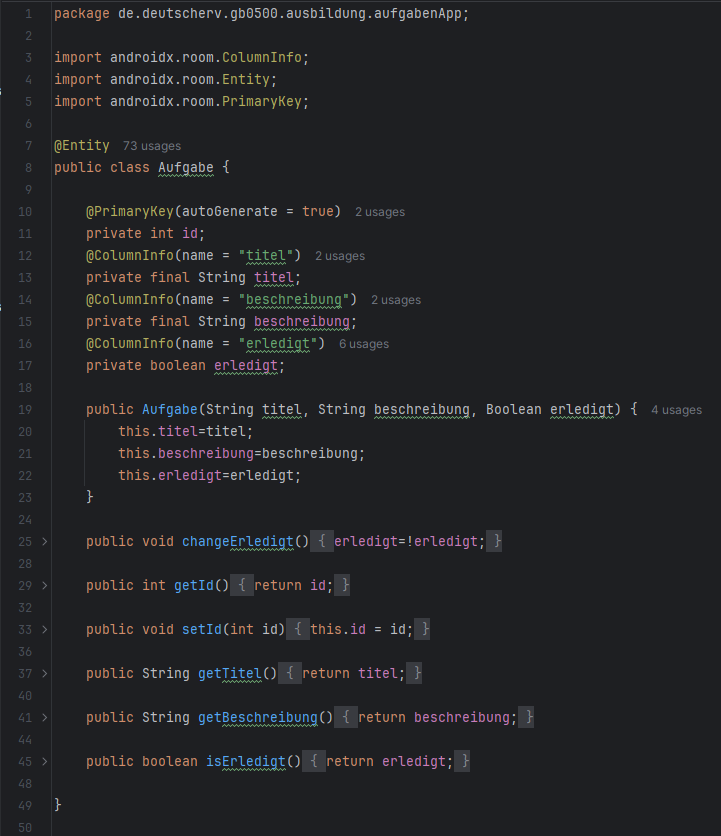
### Konstruktor

Ein Konstruktor wurde eingeführt, um Daten bei der Erstellung eines Objekts zu spezifizieren. Hier werden die Daten Titel, Beschreibung und erledigt für das Objekt festgelegt.



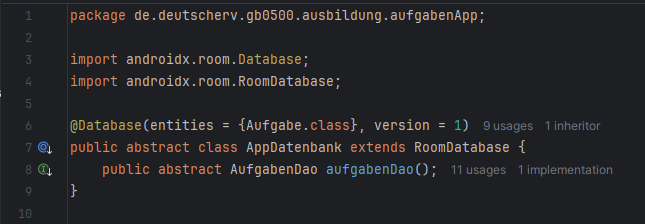
### Getter/Setter

Für die Rückgabe und Aktualisierung der Daten wurden Getter und Setter eingeführt.



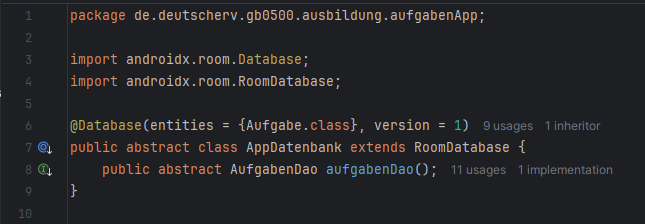
## Implementieren von Klasse AppDatenbank

In dieser Klasse muss oberhalb der Klassendeklaration @Database (entities = {[Klassenname].class}, version = 1) eingefügt werden. Dies ist notwendig, um die Klasse als Datenbank zu kennzeichnen. In entities werden die mit @Entity annotierten Klassen eingefügt. Der Parameter version enthält die Versionsnummer der Datenbank. Diese sollte erhöht werden, wenn sich das Datenbankschema ändert.



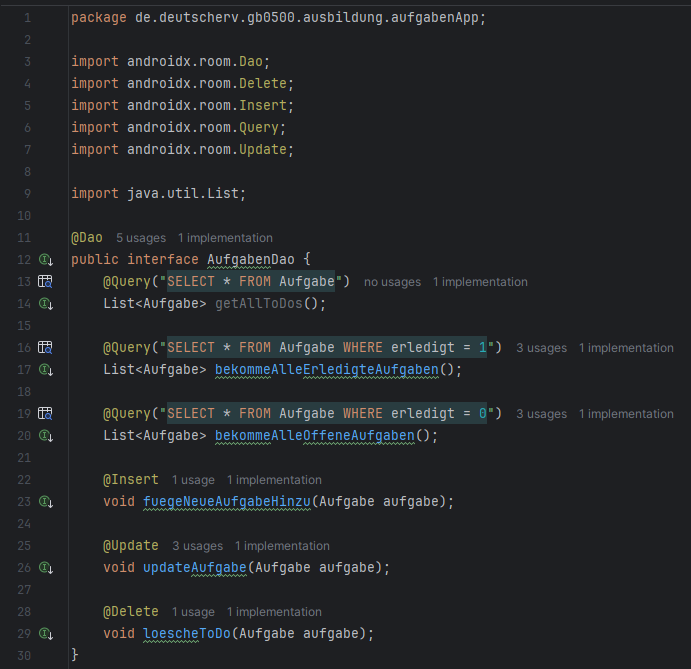
### Methode aufgabenDao

Die abstrakte Methode ist notwendig, da sie als Zugangspunkt für die Kommunikation mit der Datenbank dient. Jedes Mal, wenn etwas von der Datenbank angefordert werden soll, wird diese Methode benötigt.



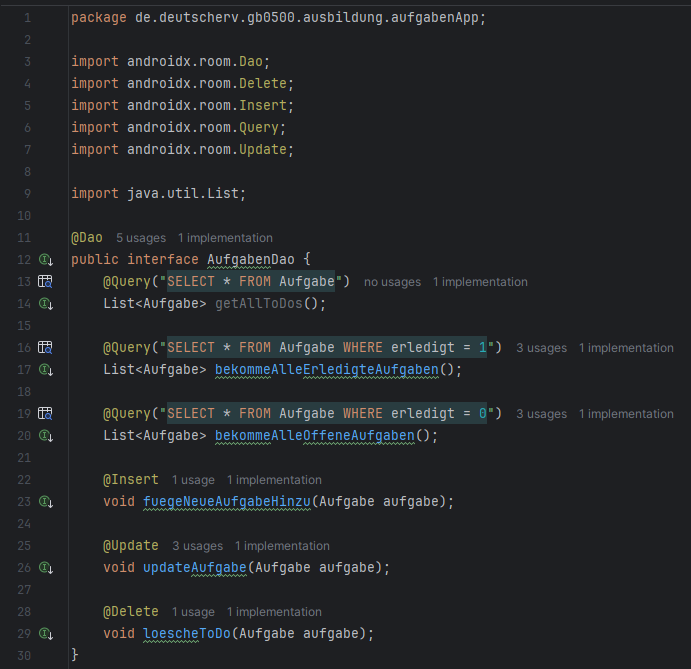
## Implementieren vom Interface AufgabenDao

Das DAO (Data Access Object) ermöglicht die Integration mit der Datenbank und das Schreiben beliebiger SQL-Abfragen. Es definiert Methoden für gängige Datenbankoperationen wie das Einfügen, Aktualisieren, Löschen und Abfragen von Daten. Um der Datenbank mitzuteilen, dass es sich bei diesem Interface (Schnittstelle) um ein DAO handelt, schreibt man @Dao über die Klassendeklaration.



### SQL-Methoden

Dies geschieht mit der @Query-Annotation. In den Klammern wird der SQL-Code in Anführungszeichen gesetzt. Die nächste Zeile enthält den Rückgabewert und den Namen der Methode. Die generierten @Insert, @Update und @Delete Annotationen erzeugen automatisch den Code, der benötigt wird, um Daten hinzuzufügen, zu aktualisieren oder zu löschen.

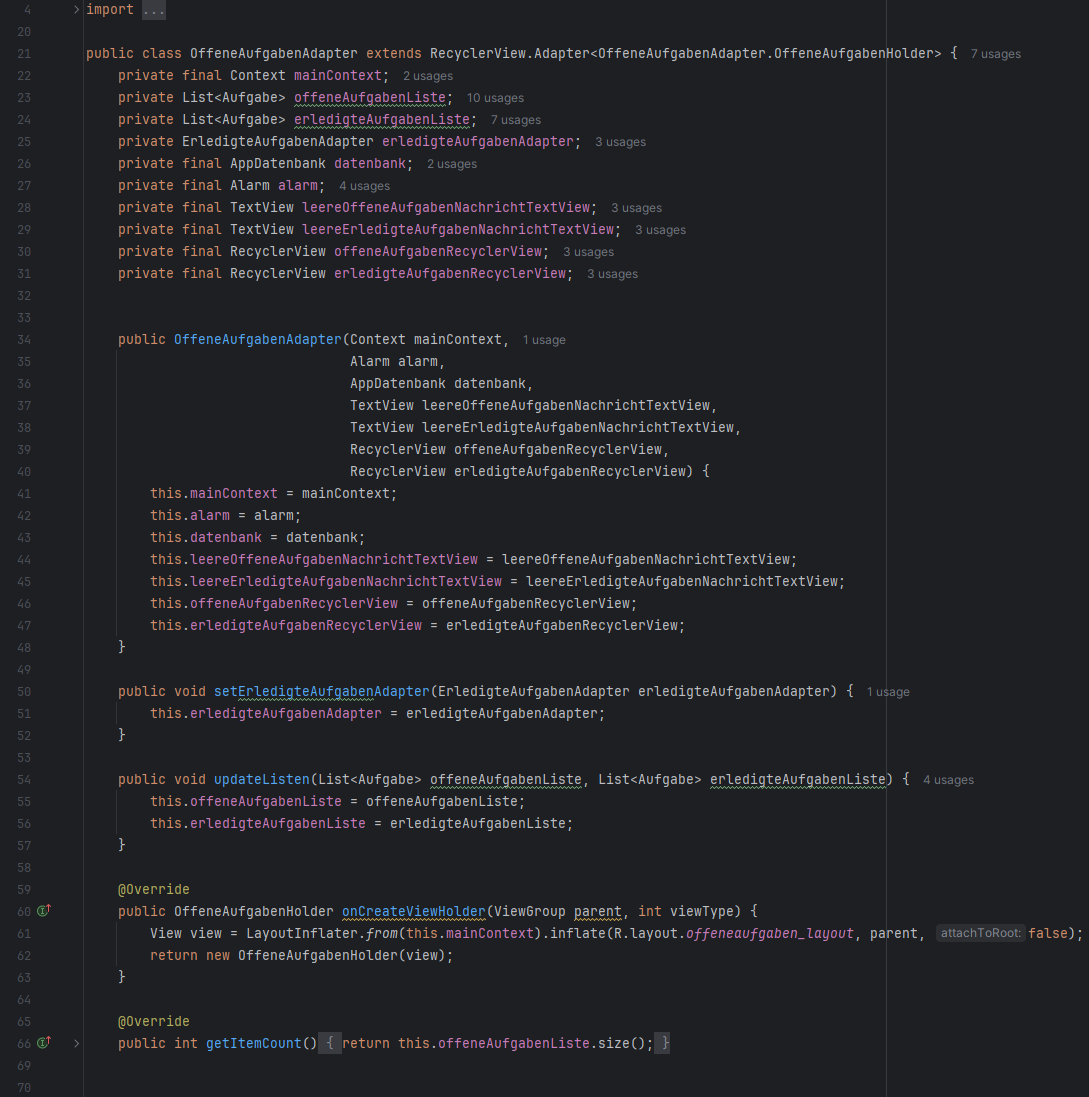


## Implementieren der Klasse OffeneAufgabenAdapter

Diese Klasse wurde erstellt, um die OffeneAufgabenRecyclerView mit den notwendigen Daten zu versorgen. Außerdem erbt sie die Klasse RecyclerView.Adapter<[Klassenname].[KlassenHolder]>.

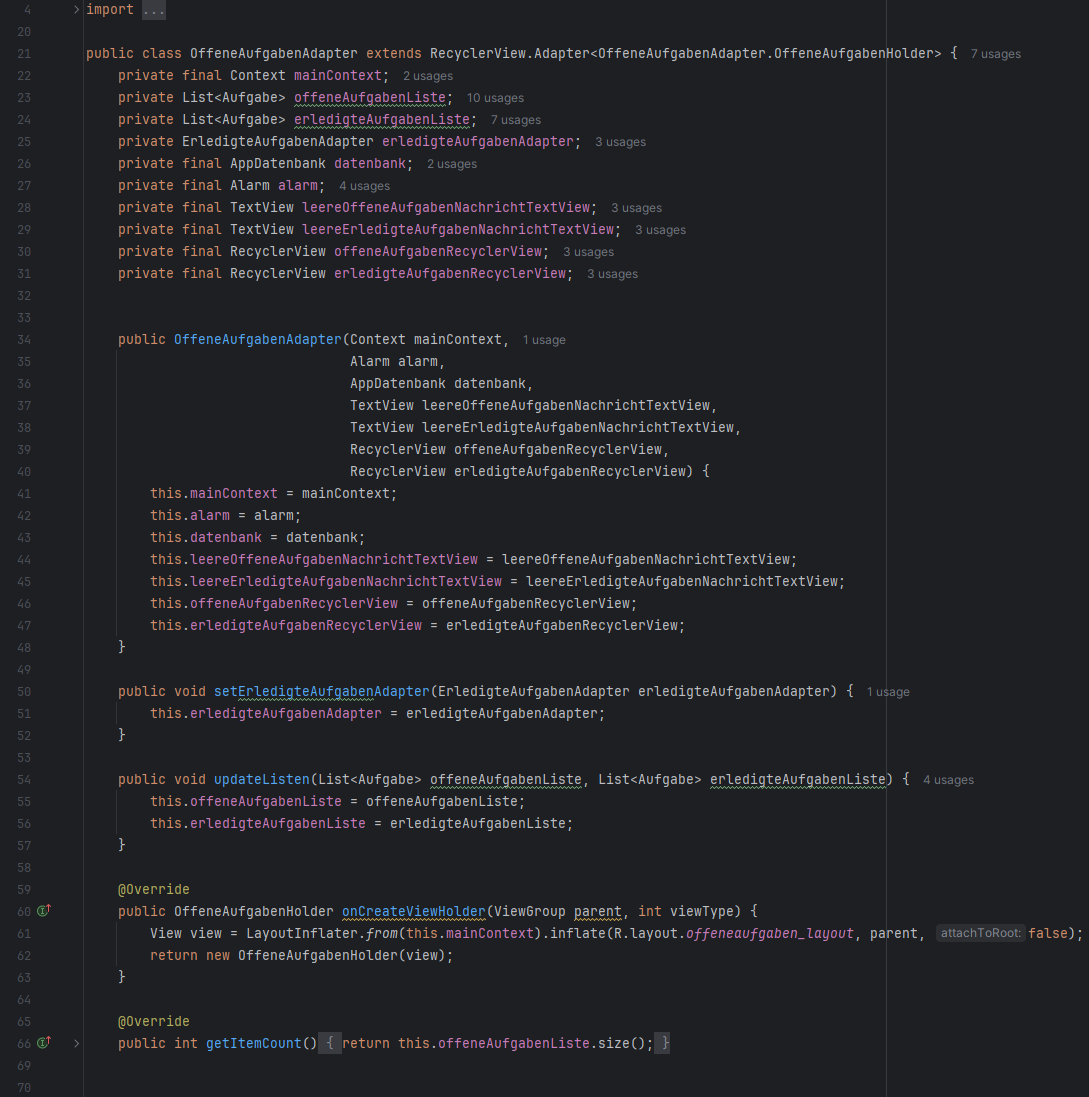
### Attribute

Die Klasse hat den Context, zwei Listen, den ErledigteAufgabenAdapter, die Datenbank, den Alarm, zwei TextViews und zwei RecyclerViews. Der Context wird für den LayoutInflater benötigt, da er den Zugriff auf Ressourcen, die Integration mit Systemdiensten und potenziell die Manipulation von Views im Zusammenhang mit der Darstellung und Handhabung von offeneAufgaben ermöglicht. Alle Aufgaben sind in zwei Listen aufgeteilt. Eine Liste enthält alle offenen Aufgaben (Aufgaben, die nicht abgeschlossen sind) und die andere Liste enthält alle erledigten Aufgaben (Aufgaben, die abgeschlossen sind). Der ErledigteAufgabenAdapter wird nur benötigt, um zu benachrichtigen, dass sich Daten geändert haben. Die Datenbank wird benötigt, um Änderungen in der Datenbank vorzunehmen. Der Alarm wird benötigt, um Popup-Fenster anzuzeigen, damit entweder Entscheidungen getroffen oder neue Daten eingegeben werden können. Die beiden TextViews und die RecyclerViews sind nur für das Design verantwortlich.



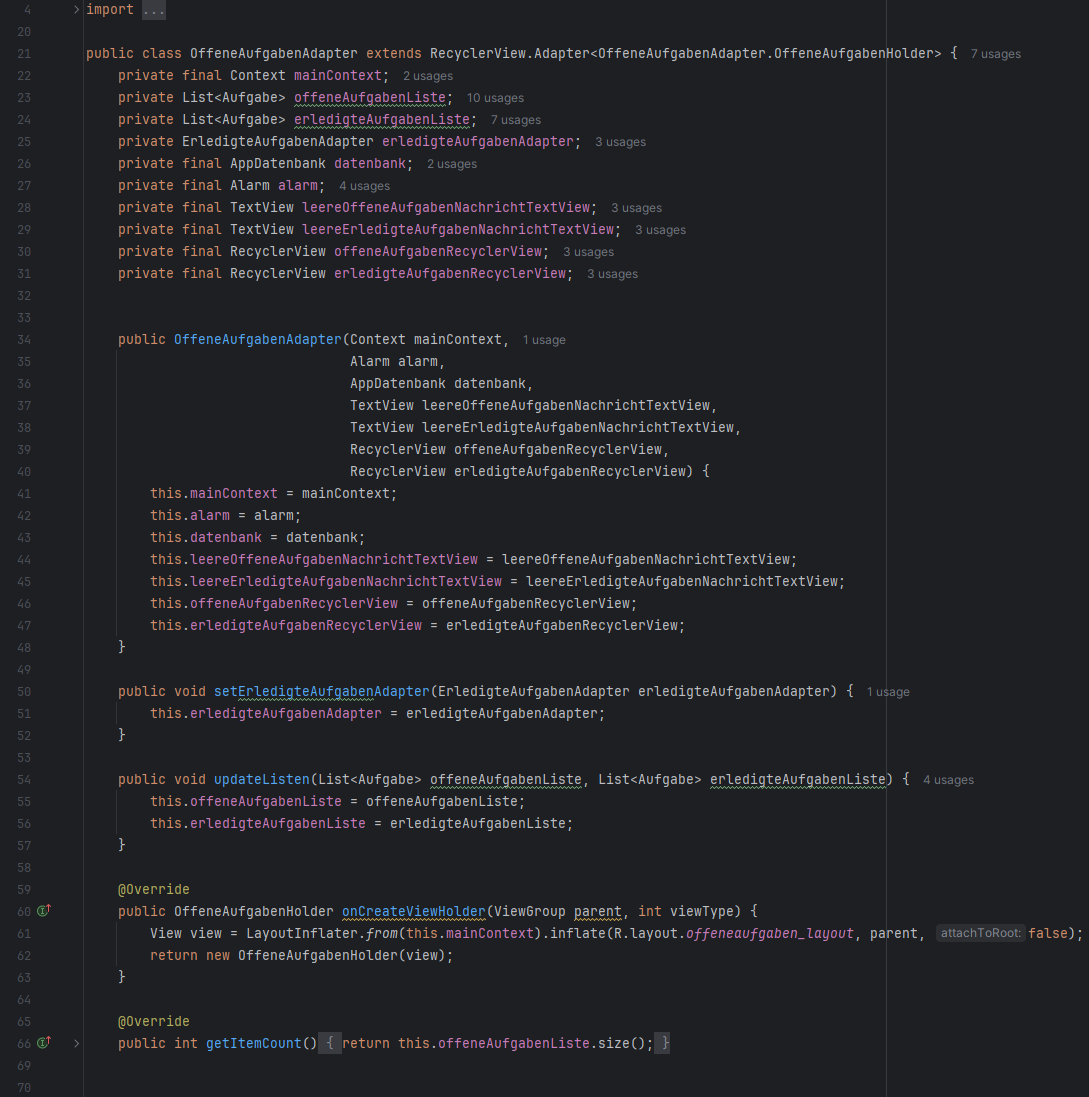
### Konstruktor

Es wurde ein Konstruktor eingeführt, um die Daten bei der Erstellung eines Objekts zu spezifizieren. Hier werden die Daten Context, Alarm, Datenbank, zwei TextViews und RecyclerViews für das Objekt festgelegt.



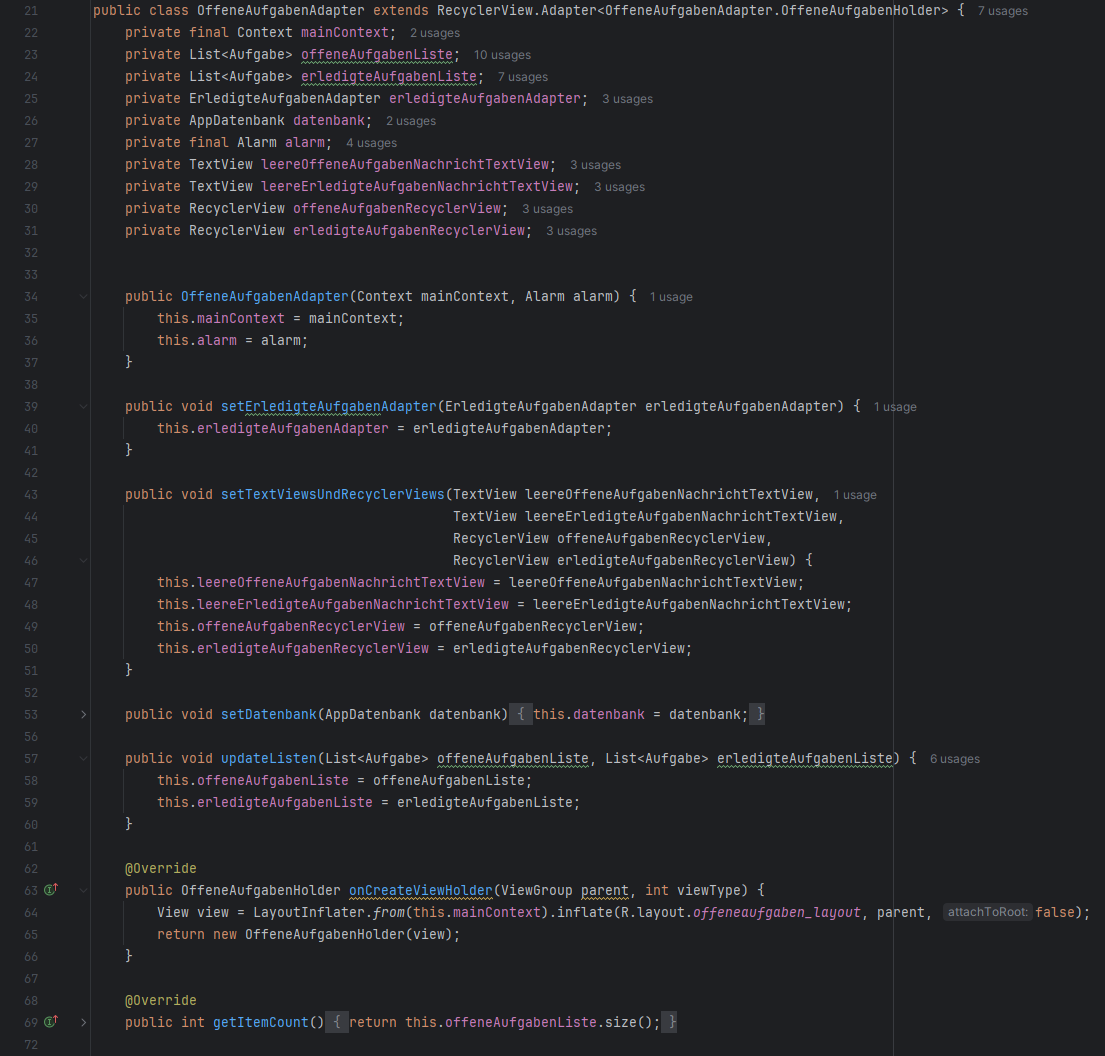
### Setter

Setter werden benötigt, um die Klasse mit Daten zu belegen.



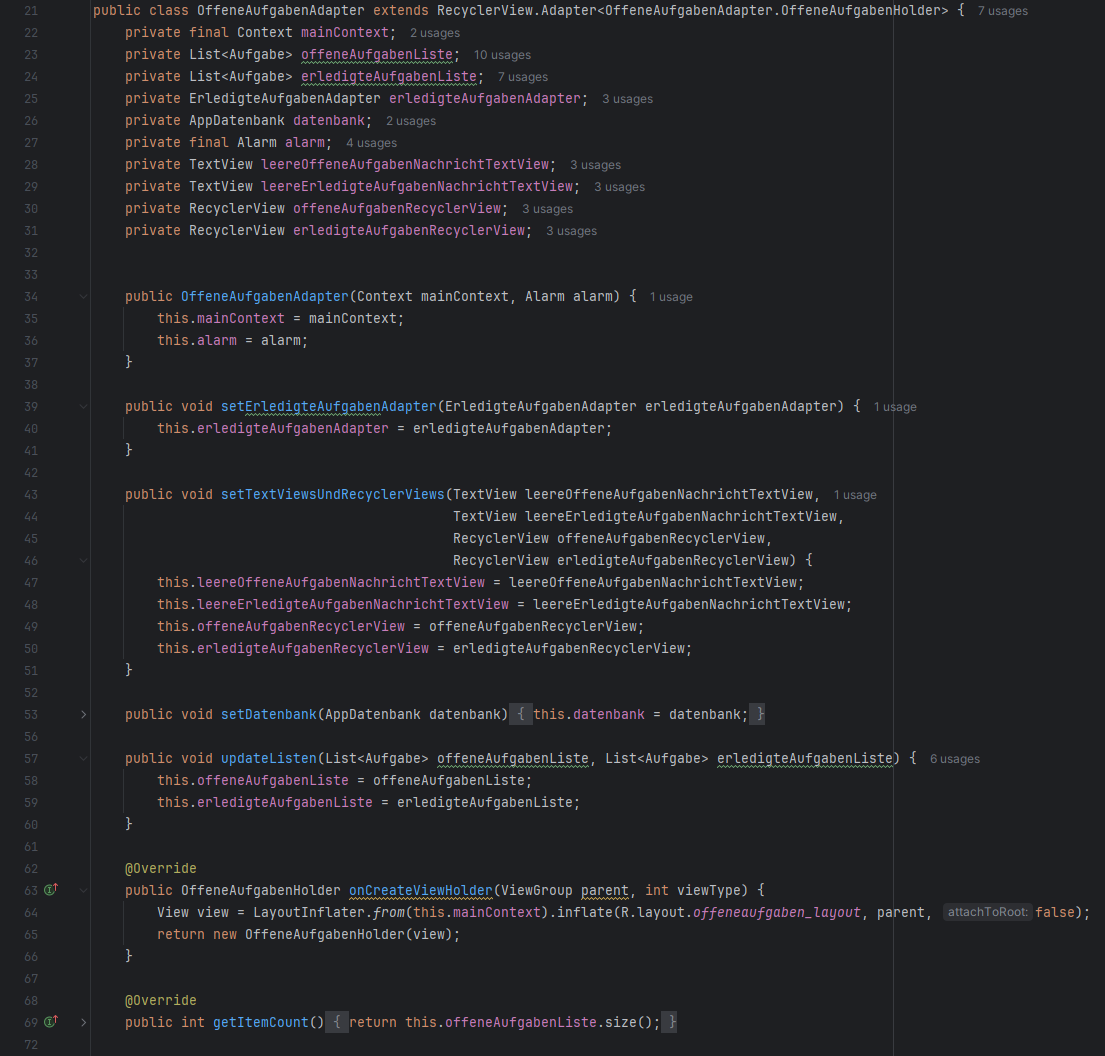
### Methode onCreateViewHolder

Diese überschriebene Methode dient zum Erstellen und Konfigurieren der View Holder, die die einzelnen Elemente in der OffeneAufgabenRecyclerView anzeigen. Das Layout der RecyclerView wird hier eingegeben.



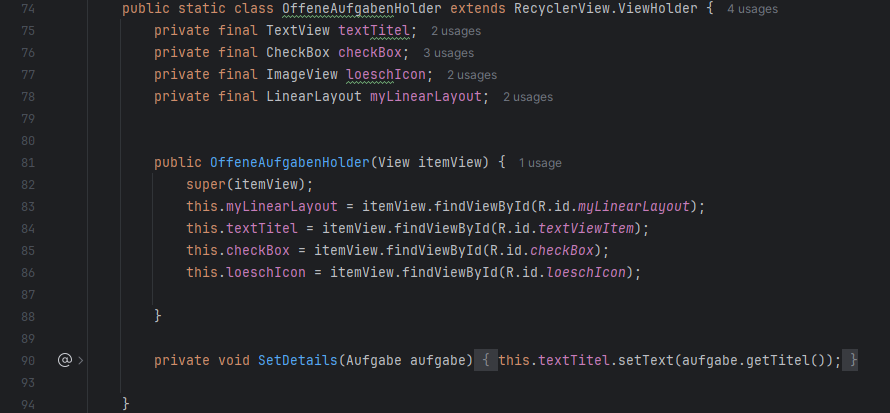
### Methode getItemCount

Diese überschriebene Methode gibt die Länge der Liste zurück.



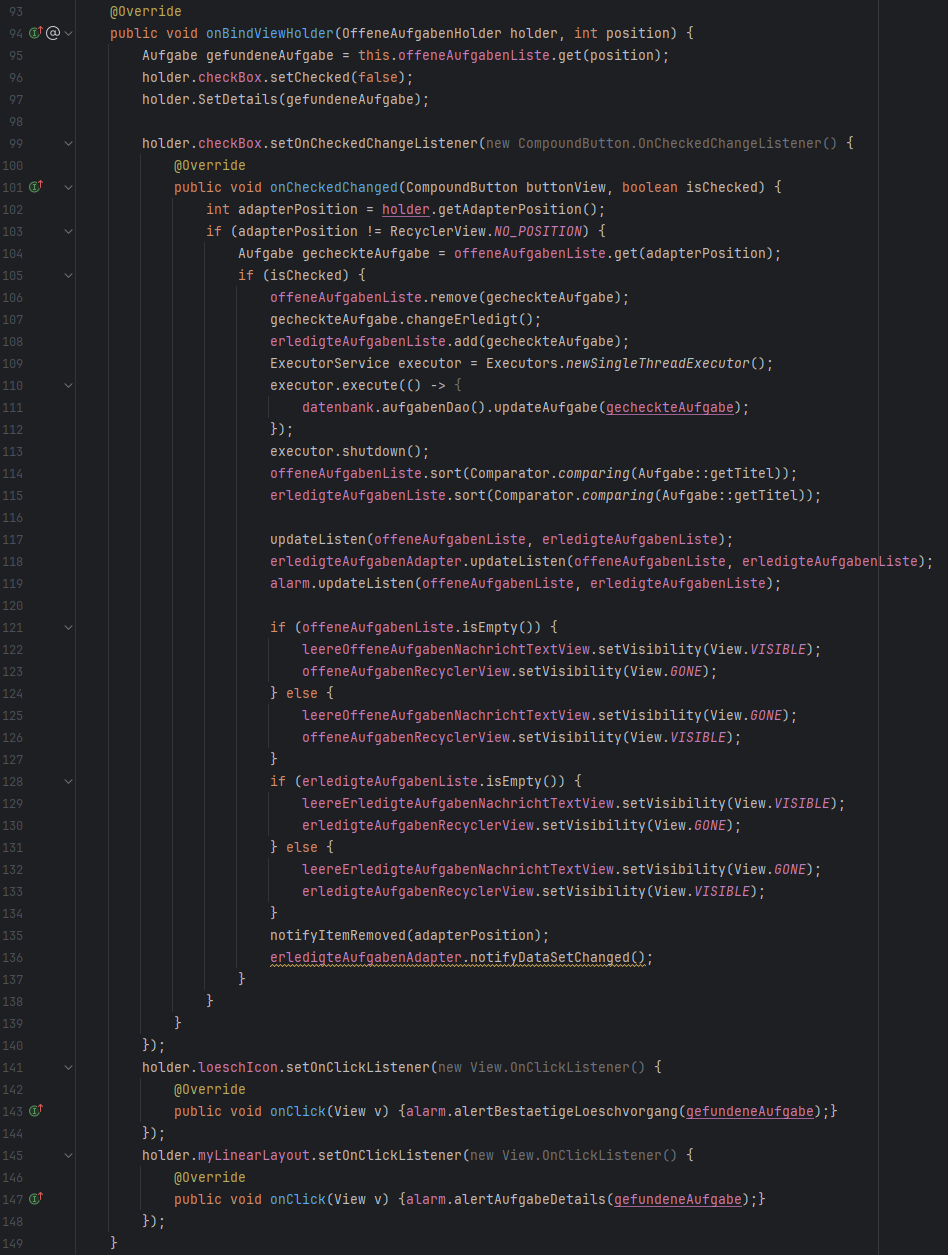
### Statische Klasse OffeneAufgabenHolder

Diese Klasse erbt die Klasse RecyclerView.ViewHolder. Dies spielt eine entscheidende Rolle für die effiziente Darstellung und Verwaltung der Daten innerhalb eines RecyclerViews. In der Klasse werden Attribute erzeugt, die beim Aufruf des OffeneAufgabenHolder verwendet werden. Die Methode [Attributname].findViewById(R.id.[IDname]) sucht im LayoutInflater nach der ID. R ist eine von Android automatisch erzeugte Klasse. Sie dient als zentraler Referenzpunkt für alle Ressourcen im Projekt. Schließlich gibt es noch eine SetDetails Methode, mit der Daten in eine View eingefügt werden können.



### Methode onBindViewHolder

In der Klasse wird Logik verwendet, um auf bestimmte Ereignisse zu reagieren. In dieser Methode wurden drei Listener definiert. Für die Checkbox, das Lösch-Icon und das Layout. Jedes Mal, wenn die Checkbox, das Lösch-Icon oder das Layout angeklickt wird, werden die Listener ausgeführt. Beim CheckboxListener ändert sich der Status der Aufgabe, wenn die Checkbox angeklickt wird. Dies wird in der Datenbank und im Code geändert. Der ExecutorService wird verwendet, um Datenbankoperationen in einem Thread auszuführen. Android erlaubt keine Operationen, die zu ANR's (Application not Responding) führen können. Dann wird die Aufgabe in die andere Liste eingefügt und die Listen werden bei den Adaptern und dem Alarm aktualisiert. Am Ende wird die Designlogik ausgeführt. Bei den anderen Listenern wird eine Methode von Alert ausgeführt.



## Implementieren der Klasse ErledigteAufgabenAdapter

Der ErledigteAufgabenAdapter ist dem OffeneAufgabenAdapter sehr ähnlich, macht aber meistens das Gegenteil der Logik. Diese Klasse wurde erstellt, um die ErledigteAufgabenRecyclerView mit den notwendigen Daten zu versorgen. Sie erbt auch die Klasse RecyclerView.Adapter<[Klassenname].[KlassenHolder]>.

### Attribute

Die Klasse hat den Context, zwei Listen, den OffeneAufgabenAdapter, die Datenbank, den Alarm, zwei TextViews und zwei RecyclerViews. Der Context wird für den LayoutInflater benötigt, da er den Zugriff auf Ressourcen, die Integration mit Systemdiensten und potenziell die Manipulation von Views im Zusammenhang mit der Darstellung und Handhabung von offeneAufgaben ermöglicht. Alle Aufgaben sind in zwei Listen unterteilt. Die erste Liste enthält alle offenen Aufgaben (Aufgaben, die noch nicht abgeschlossen sind) und die zweite Liste enthält alle erledigten Aufgaben (Aufgaben, die abgeschlossen sind). Der OffeneAufgabenAdapter wird nur benötigt, um zu benachrichtigen, dass sich Daten geändert haben. Die Datenbank wird benötigt, um Änderungen sowohl in der Anwendung als auch in der Datenbank vorzunehmen. Der Alarm wird benötigt, um Popup-Fenster anzuzeigen, damit entweder Entscheidungen getroffen oder neue Daten eingegeben werden können. Die beiden TextViews und die RecyclerViews sind nur für das Design verantwortlich.



### Konstruktor

Es wurde ein Konstruktor eingeführt, um die Daten bei der Erstellung eines Objekts zu spezifizieren. Hier werden die Daten Context, OffeneAufgabenAdapter, Alarm, Datenbank, zwei TextViews und zwei RecyclerViews für das Objekt festgelegt.



### Methode updateListen

 Diese Methode aktualisiert die Listen in der Klasse.

### Methode onCreateViewHolder

Diese überschriebene Methode dient zum Erstellen und Konfigurieren der View Holder, die die einzelnen Elemente in der ErledigteAufgabenRecyclerView anzeigen. Hier wird das Layout der RecyclerView angegeben.

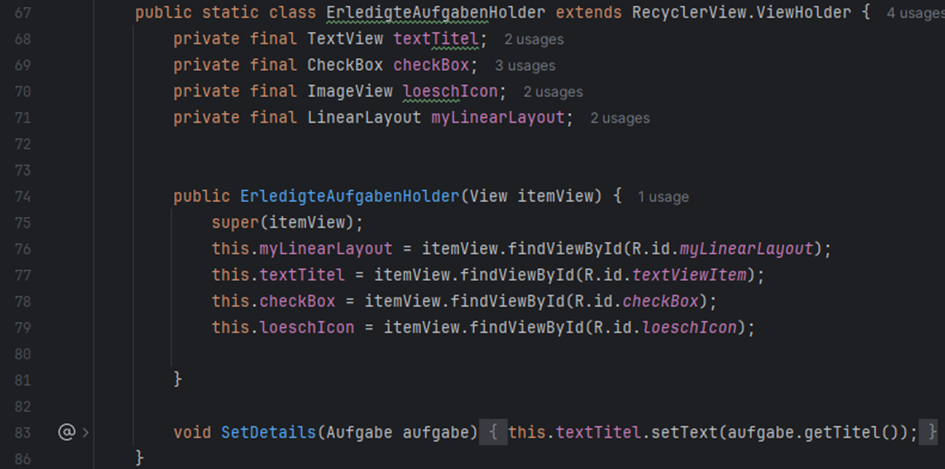


### Methode getItemCount

Diese überschriebene Methode gibt die Länge der Liste zurück.



### Statische Klasse ErledigteAufgabenHolder

Diese Klasse erbt die Klasse RecyclerView.ViewHolder. Dies spielt eine entscheidende Rolle für die effiziente Darstellung und Verwaltung der Daten innerhalb eines RecyclerViews. In der Klasse werden Attribute erzeugt, die beim Aufruf von ErledigteAufgabenHolder verwendet werden. Die Methode [Attributname].findViewById(R.id.[IDname]) sucht im LayoutInflater nach der ID. R ist eine von Android automatisch erzeugte Klasse. Sie dient als zentraler Referenzpunkt für alle Ressourcen im Projekt. Schließlich gibt es noch eine SetDetails Methode, mit der Daten in eine View eingefügt werden können.

### Methode onBindViewHolder

In der Klasse wird Logik verwendet, um auf bestimmte Ereignisse zu reagieren. In dieser Methode wurden drei Listener definiert. Für die Checkbox, das Lösch-Icon und das Layout. Jedes Mal, wenn die Checkbox, das Lösch-Icon oder das Layout angeklickt wird, werden die Listener ausgeführt. Beim CheckboxListener ändert sich der Status der Aufgabe, wenn die Checkbox angeklickt wird. Dies wird in der Datenbank und im Code geändert. Der ExecutorService wird verwendet, um Datenbankoperationen in einem Thread auszuführen. Android erlaubt keine Operationen, die zu ANR's (Application not Responding) führen können. Dann wird die Aufgabe in die andere Liste eingefügt und die Listen werden bei den Adaptern und dem Alarm aktualisiert. Am Ende wird die Designlogik ausgeführt. Bei den anderen Listenern wird eine Methode von Alarm ausgeführt.

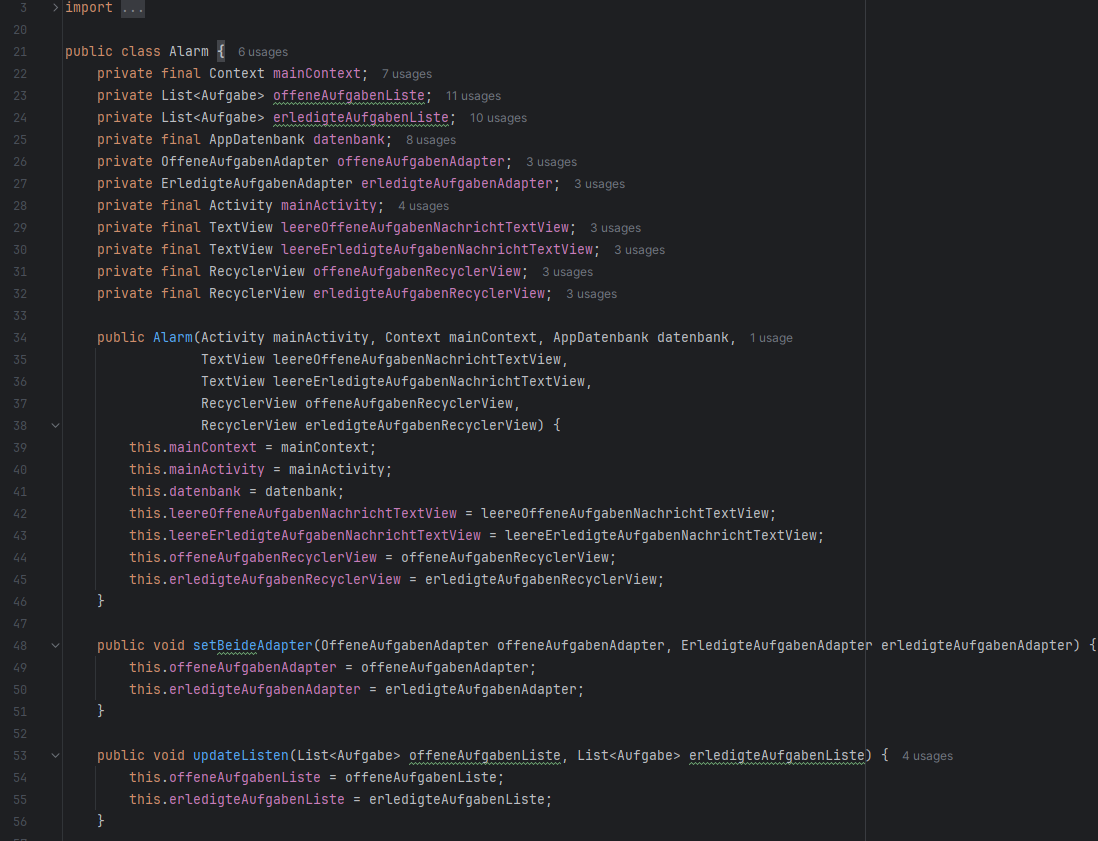


## Implementieren der Klasse Alarm

Die Klasse ist für die Verwaltung von Warndialogen zuständig. Dies sind Popup-Fenster, die erscheinen, um den Benutzer zu informieren, eine Aktion zu bestätigen oder Eingaben zu sammeln.

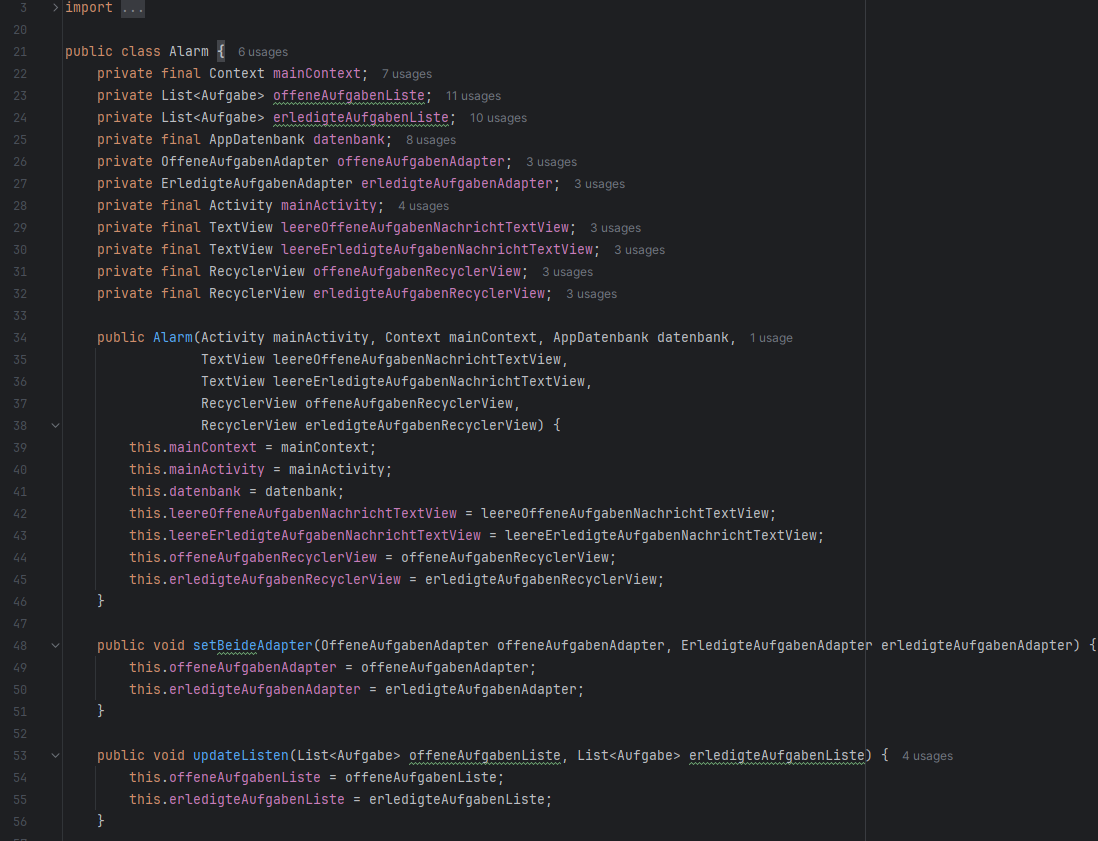
### Attribute

Die Klasse hat den Context, zwei Listen, die Datenbank, den Erledigte- und OffeneAufgabenAdapter, die Aktivität, zwei TextViews und zwei RecyclerViews. Der Context wird für den AlertBuilder benötigt, da er den Zugriff auf Ressourcen, die Integration mit Systemdiensten und potenziell die Manipulation von Views im Zusammenhang mit der Darstellung und Handhabung von offeneAufgaben ermöglicht. Alle Aufgaben sind in zwei Listen unterteilt. Eine Liste enthält alle offeneAufgaben (Aufgaben, die nicht abgeschlossen sind) und die andere Liste enthält alle erledigteAufgaben (Aufgaben, die abgeschlossen sind). Der Erledigte und der OffeneAufgabenAdapter werden nur benötigt, um mitzuteilen, dass sich Daten geändert haben. Die Datenbank wird benötigt, um Änderungen in der Datenbank vorzunehmen. Die Aktivität wird benötigt, um vom Thread auf die Main-Methode zuzugreifen und dort das Design (die beiden TextViews und die RecyclerViews) zu ändern. Die beiden TextViews und die RecyclerViews sind nur für das Design zuständig.



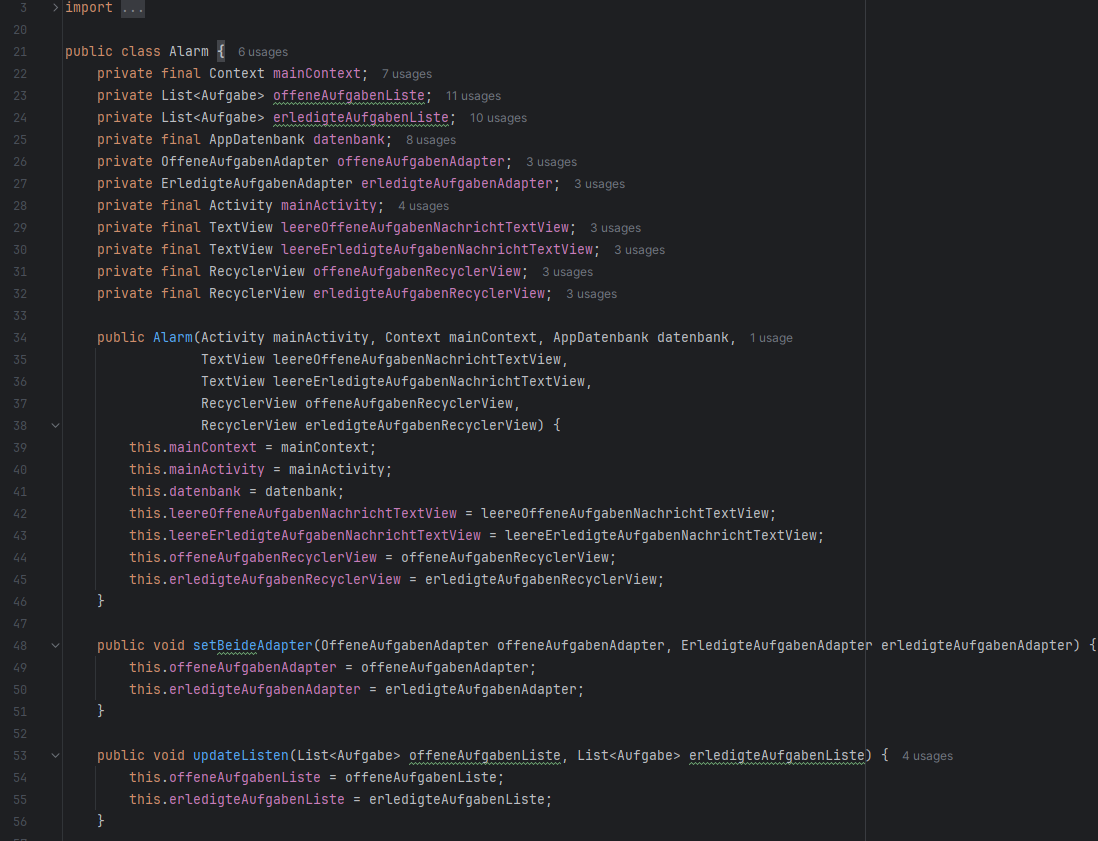
### Konstruktor

Es wurde ein Konstruktor eingeführt, um Daten bei der Erstellung eines Objektes zu spezifizieren. Hier werden die Daten Context, Aktivität, Datenbank, zwei TextViews und zwei RecyclerViews für das Objekt festgelegt.



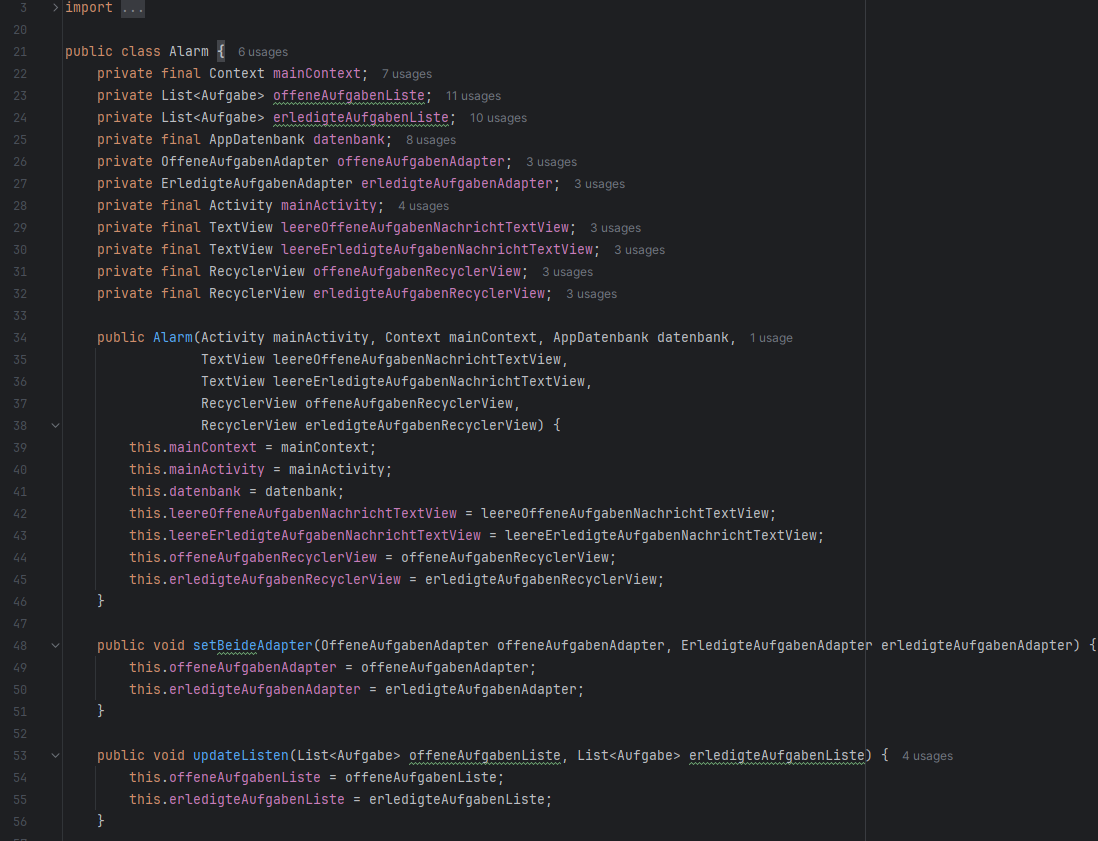
### Setter

setBeideAdapter wird benötigt, um die Adapter an die Klasse zu übergeben, um die Adapter zu informieren, dass sich Daten geändert haben.



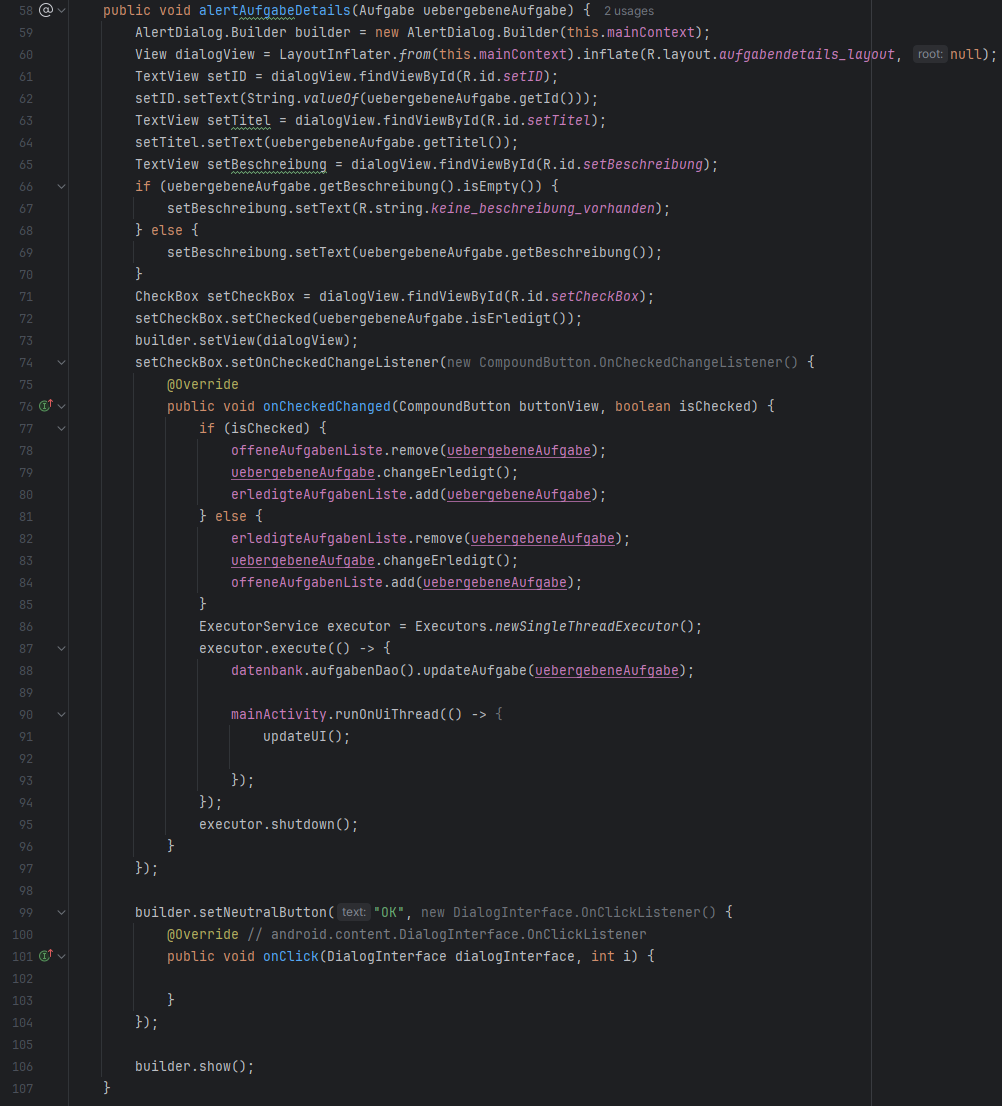
### Methode updateListen

updateListen wird verwendet, um die Listen zu aktualisieren, so dass immer aktuelle Daten zur Verfügung stehen und nicht veraltet sind.



### Methode alertAufgabeDetails

Diese Methode wurde erstellt, um die Details der Aufgabe anzuzeigen. Details wie ID, Titel, Beschreibung und boolean erledigt werden in einem AlertDialog angezeigt. Wenn die Beschreibung leer ist, wird (keine Beschreibung vorhanden) angezeigt. In dieser View kann der Status erledigt geändert werden. Ansonsten gibt es einen Button, um die View zu schließen.



Der Dialog sieht folgendermaßen aus:

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

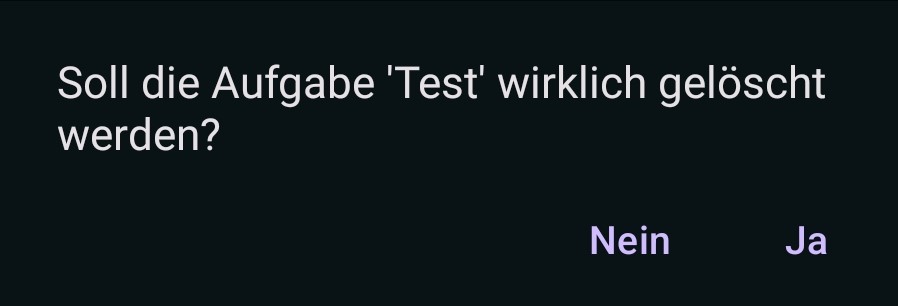
Automatisch generierte Beschreibung

### Methode alertBestaetigeLoeschvorgang

Diese Methode wurde entwickelt, um dem Benutzer die Möglichkeit zu geben, mit Hilfe eines Popup-Fensters eine Entscheidung zu treffen. Es stehen eine Meldung und zwei Knöpfe zur Verfügung. Der Benutzer muss entscheiden, ob er die Aufgabe wirklich löschen oder behalten möchte.



Der Dialog sieht folgendermaßen aus:

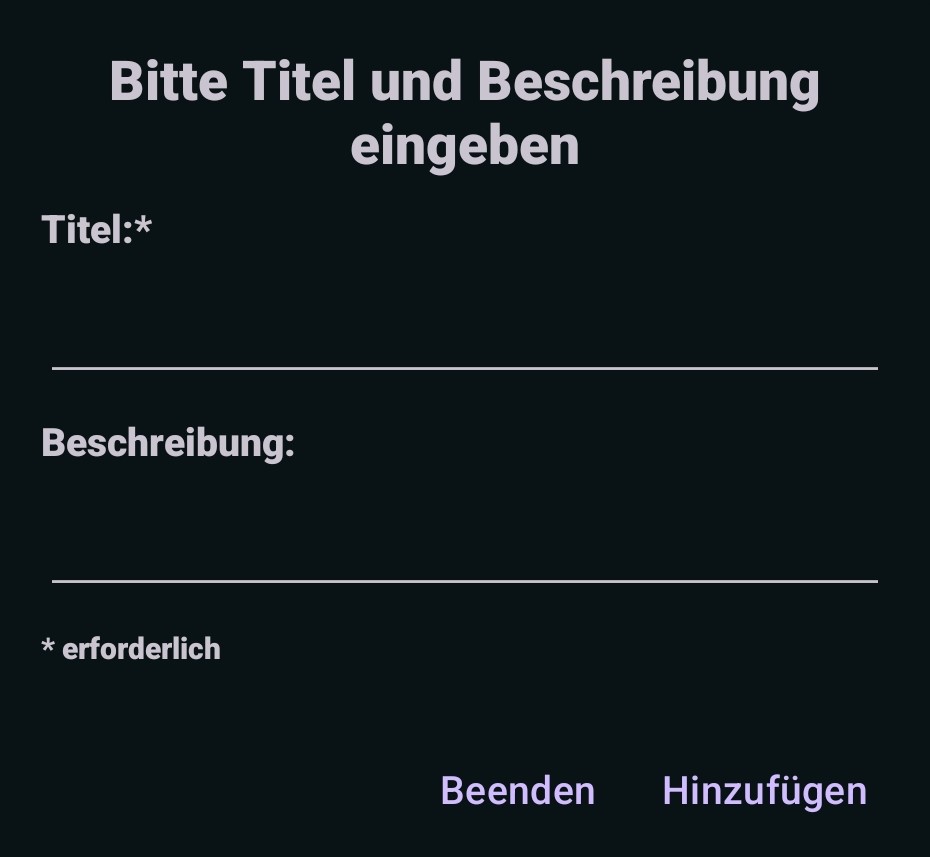


### Methode alertNeueAufgabeErstellen

Dieser AlertDialog erstellt mit Hilfe eines Layouts zwei Eingabefenster, in die Daten eingegeben werden können. Dieser Alert ist für die Erstellung neuer Aufgaben vorgesehen. Er fragt nach dem Titel und der Beschreibung. Der Titel ist ein Pflichtfeld. Daher wird überprüft, ob dieses Feld leer ist. Wenn es leer ist, wird eine weitere Warnung alertEmptyTitel() ausgegeben. Wenn das Feld gefüllt ist, wird die Erstellung des Objekts erlaubt. Wenn das Beschreibungsfeld leer ist, wird es leer zurückgegeben und das Objekt wird erstellt. Wenn eine neue Aufgabe erstellt wird, wird ihr boolean erledigt immer auf false gesetzt.

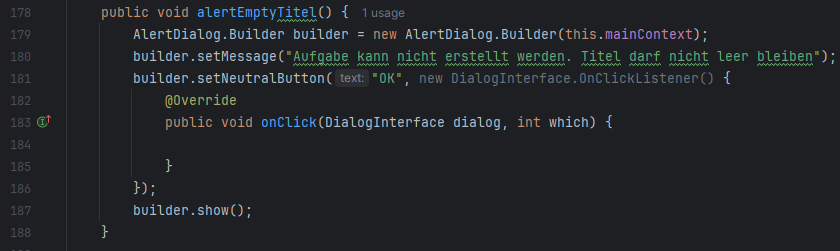


Der Dialog sieht folgendermaßen aus:



### Methode alertEmptyTitel

Diese Methode wird verwendet, um eine Warnung auszugeben. Es soll nur angezeigt werden, dass die Aufgabe nicht erstellt werden konnte, da der Titel leer ist.



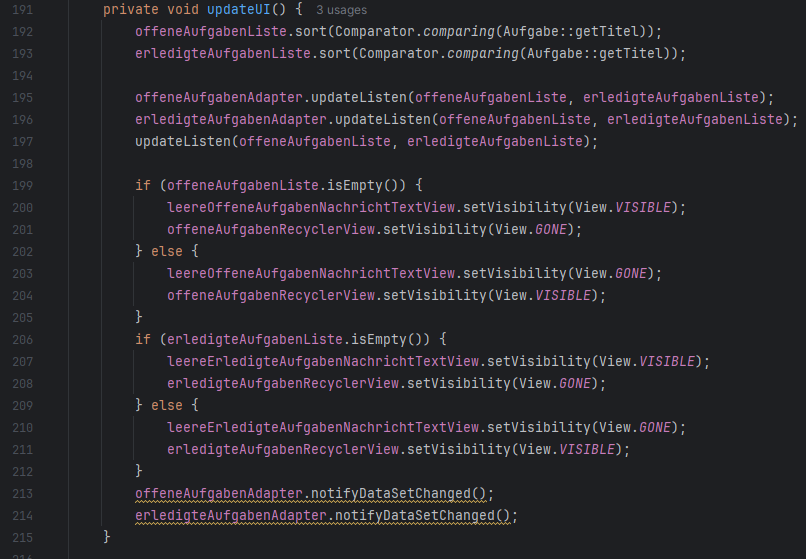
Der Dialog sieht folgendermaßen aus:

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

### Methode updateUI

Diese Methode wurde nur erstellt, um die Listen zu sortieren, sie für die Adapter zu aktualisieren und die RecyclerViews unsichtbar zu machen, wenn sie leer sind. An der gleichen Stelle erscheint ein TextView, das nur anzeigt, dass es keine Aufgaben gibt. Am Ende informiert es die Adapter, dass die Listen aktualisiert wurden.

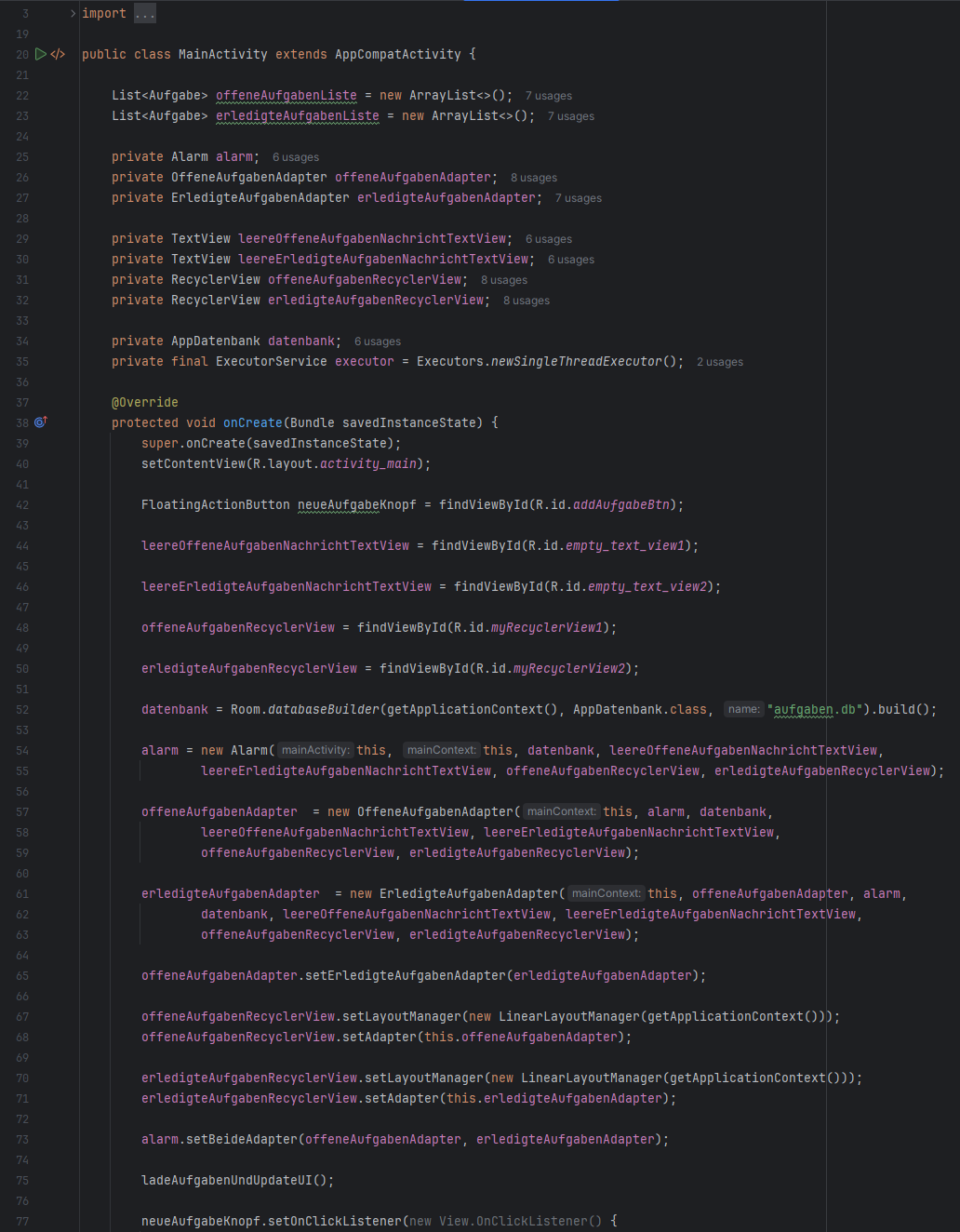


## Implementieren der Klasse MainActivity

Diese Klasse wird immer automatisch generiert, wenn ein neues Projekt erstellt wird. Diese Klasse wird beim Starten der Anwendung ausgeführt.

### Attribute

Diese Klasse enthält die beiden Listen für die offenen und die erledigten Aufgaben. Außerdem besitzt sie den Alarm, den OffeneAufgabenAdapter und den ErledigteAufgabenAdapter sowie die Datenbank, den Executor, die beiden TextViews und die RecyclerViews. Diese werden in der Methode onCreate (Main-Methode) erzeugt.



### Methode onCreate

Diese Methode wird automatisch von Android erzeugt. In dieser Methode wird die Applikation gestartet. Hier werden die TextViews und die RecyclerViews zugewiesen, sowie der Alarm, der Offene und der ErledigteAufgabenAdapter, die Datenbank und ein FloatingActionButton. Der Button ist für das Anlegen neuer Aufgaben zuständig. Ihm wurde ein onClickListener zugewiesen. Wenn dieser Button angeklickt wird, wird der alert alertNeueAufgabeErstellen() ausgeführt. Die Setter von Alert, Offene- und ErledigteAufgabeAdapter werden verwendet, um alle notwendigen Daten/Objekte zu setzen.

Ein Bild, das Text, Screenshot, Karte Menü enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Karte Menü enthält.

Automatisch generierte Beschreibung

### Methode ladeAufgabenUndUpdateUI

Diese Methode startet einen Executor (Thread). Dieser führt dann leseDatenbankUndAktualisierListen aus. Im runOnUiThread wird die Methode AktualisiereViewSichtbarkeit ausgeführt und die Adapter werden benachrichtigt, dass sich Daten geändert haben.

Ein Bild, das Text, Screenshot, Karte Menü enthält.

Automatisch generierte Beschreibung

### Methode leseDatenbankUndAktualisierListen

Diese Methode liest die Daten aus der Datenbank, fügt die Daten in die Liste ein, sortiert die Liste und aktualisiert beide Listen und den Alert.



### Methode aktualisiereViewSichtbarkeit

Diese Methode ist nur für das Design. Sie sorgt dafür, dass die RecyclerView verschwindet und durch eine TextView ersetzt wird, wenn die Liste leer ist, und zwar sowohl für die OffeneAufgabenRecyclerView als auch für die ErledigteAufgabenRecyclerView.



## Ressourcen

Ressourcen werden in Android verwendet, um das Layout, das App-Icon, Farben und Themes zu ändern. Alles wird im Ordner res gespeichert. In diesem Projekt wurden diese Dateien modifiziert.

### Drawable

Das Logo wurde mit Hilfe von Android Studio erstellt, da für das App-Icon nur XML verwendet wird. In Android Studio kann man mit Hilfe von Vector Asset eine SVG in XML konvertieren. Unter File -> New -> Vector Asset. Aus der SVG-Datei wird eine XML-Datei generiert. Mit dieser kann man den XML-Code in ic\_launcher\_foreground einfügen. Der Hintergrund (ic\_launcher\_background) ist nur weiß. Außerdem wurde das Logo als PNG gespeichert, um es im Startmenü anzeigen zu können.

### Layouts

Das Layout ist frei wählbar. Die beiden RecyclerViews, in denen die Aufgaben angezeigt werden, sind die wichtigsten Elemente dieses Projekts. Außerdem wurde ein neues Attribut (itemBackgroundColor) vom Format Color erstellt, mit dem die Hintergrundfarbe des Items mit Hilfe der Themes geändert werden kann.

### Values

Im Ordner Values befinden sich die XML-Dateien mit den Attributen, den Colors, den Strings und den Themes. In der Attributes Datei wurde nur ein Attribut angelegt, um beim Wechsel in den Darkmode auch die Farbe des Items im RecyclerView zu ändern. Die Strings Datei ist nicht unbedingt notwendig. In dieser Datei werden nur hartcodierte Textstrings durch kurze Referenzen ersetzt, die im Code verwendet werden. Die Colors-Datei ist der Strings-Datei ähnlich. Sie verwendet ebenfalls Referenzen, jedoch auf einen Farbcode. Dadurch muss man sich nicht mehr den Farbcode merken, sondern kann einfach die Referenz verwenden. Die Themes Dateien in Android sind für die Definition des allgemeinen Aussehens der App verantwortlich. Sie steuern das Styling und das Aussehen verschiedener UI-Elemente.

# Nützliche Links

[Dynamische Listen mit RecyclerView erstellen  |  Views  |  Android Developers](https://developer.android.com/develop/ui/views/layout/recyclerview?hl=de#java)

[RecyclerView.Adapter  |  Android Developers](https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.Adapter#onBindViewHolder(VH,int,java.util.List%3Cjava.lang.Object%3E))

[RecyclerView.ViewHolder  |  Android Developers](https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.ViewHolder)

[Daten mit Room in einer lokalen Datenbank speichern  |  Android Developers](https://developer.android.com/training/data-storage/room?hl=de#java)

[Dialogfelder  |  Views  |  Android Developers](https://developer.android.com/develop/ui/views/components/dialogs?hl=de)

[Stile und Designs  |  Views  |  Android Developers](https://developer.android.com/develop/ui/views/theming/themes?hl=de)

# Glossar

|  |  |  |
| --- | --- | --- |
| A | Adapter | Eine Schnittstelle zwischen einer Datenquelle und einer UI-Komponente (z.B. RecyclerView), die die Darstellung von Daten steuert. |
|  | AlertDialog | Ein Dialogfenster, das Benutzern wichtige Nachrichten anzeigt und sie zu Aktionen auffordert. |
| D | DAO | Data Access Object = Ein Objekt, welches für den Zugriff auf Datenquellen erstellt wird. Es ist typischer Weise austauschbar. |
| E | Entity | In Room ist ein Entity eine Java-Klasse, die als Tabelle in der SQLite-Datenbank fungiert. |
| J | Java | Java ist eine Objektorientierte Programmiersprache und eine eingetragene Marke des Unternehmens Sun Microsystems, welches 2010 von Oracle aufgekauft wurde. |
| R | RecyclerView | Eine RecyclerView ist ein felxibles und effizientes View-Widget in Android, das Daten in Form einer scrollbaren Liste darstellt. |
|  | Room | Eine Abstraktionsschicht über SQLite, die das Arbeiten mit Datenbanken in Android erleichtert. |
| S | SQLite | Eine relationale Datenbank, die in Android zum Speichern von Daten lokal auf dem Gerät verwendet wird. |
| T | TextView | Eine TextView ist ein einfaches UI-Element in Android. Das Texte anzeigt. |
|  | Thread | Ein Thread ist eine parallele Ausführungsreihe bei der Programmausführung. Sie beeinträchtigt die Anwendung nicht. |
| W | Workspace | Ein Workspace bezeichnet die Arbeitsumgebung in der Entwicklungsumgebung. |