To-Do-List

Appdokumentation

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

Autor: Daniel Ermilov

Inhaltsverzeichnis

1 Einleitung

1.1 Ziel und Zweck

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

1.2 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.

2 Überblick

2.1 Systemarchitektur

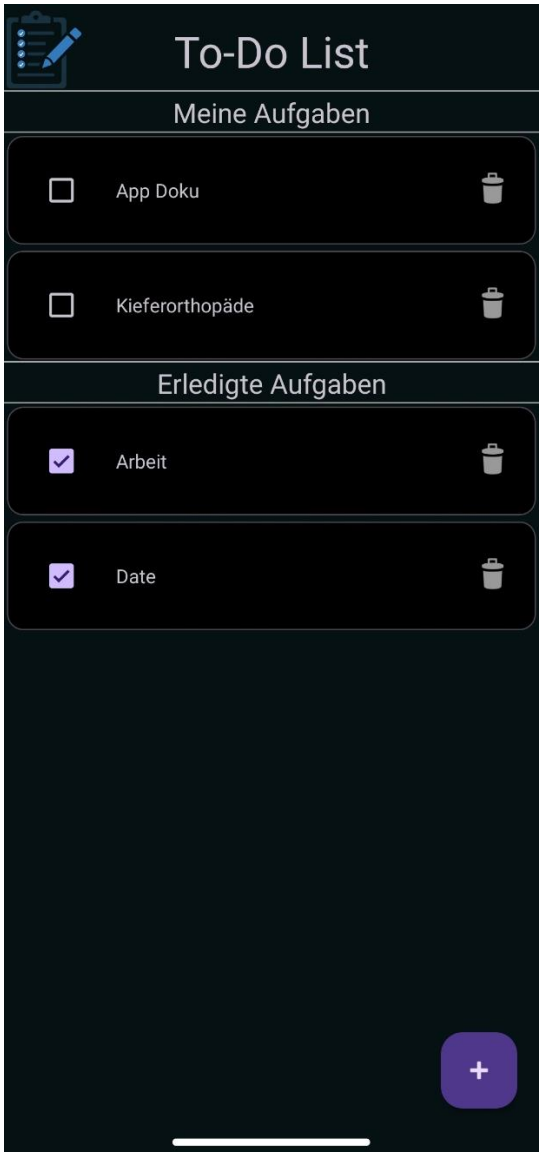
Die To-Do App nutzt die in Android integrierte Root-Datenbank (SQLite), um Aufgaben lokal auf dem Gerät des Benutzers zu speichern. Diese Datenbank ermöglicht es der App, die Daten dauerhaft zu speichern, so dass die Aufgaben auch nach dem Schließen oder Neustarten der App verfügbar bleiben.

2.2 Programmiersprachen

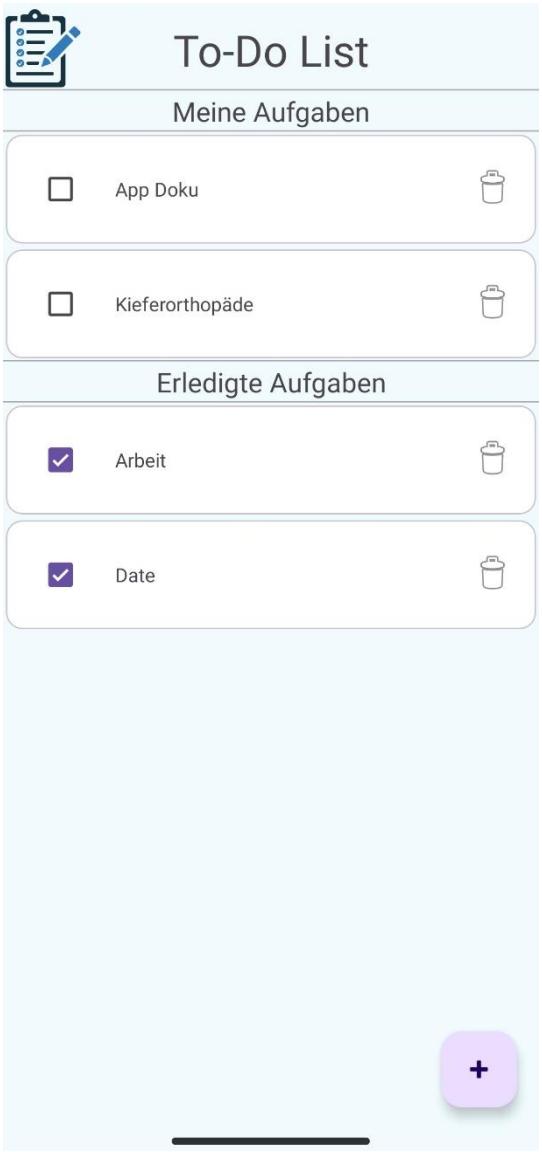
Die To-Do App wurde in Java programmiert und verwendet XML für das Design der Benutzeroberfläche. SQL wird verwendet, um mit der Datenbank zu interagieren und Daten abzufragen oder zu ändern.

2.3 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



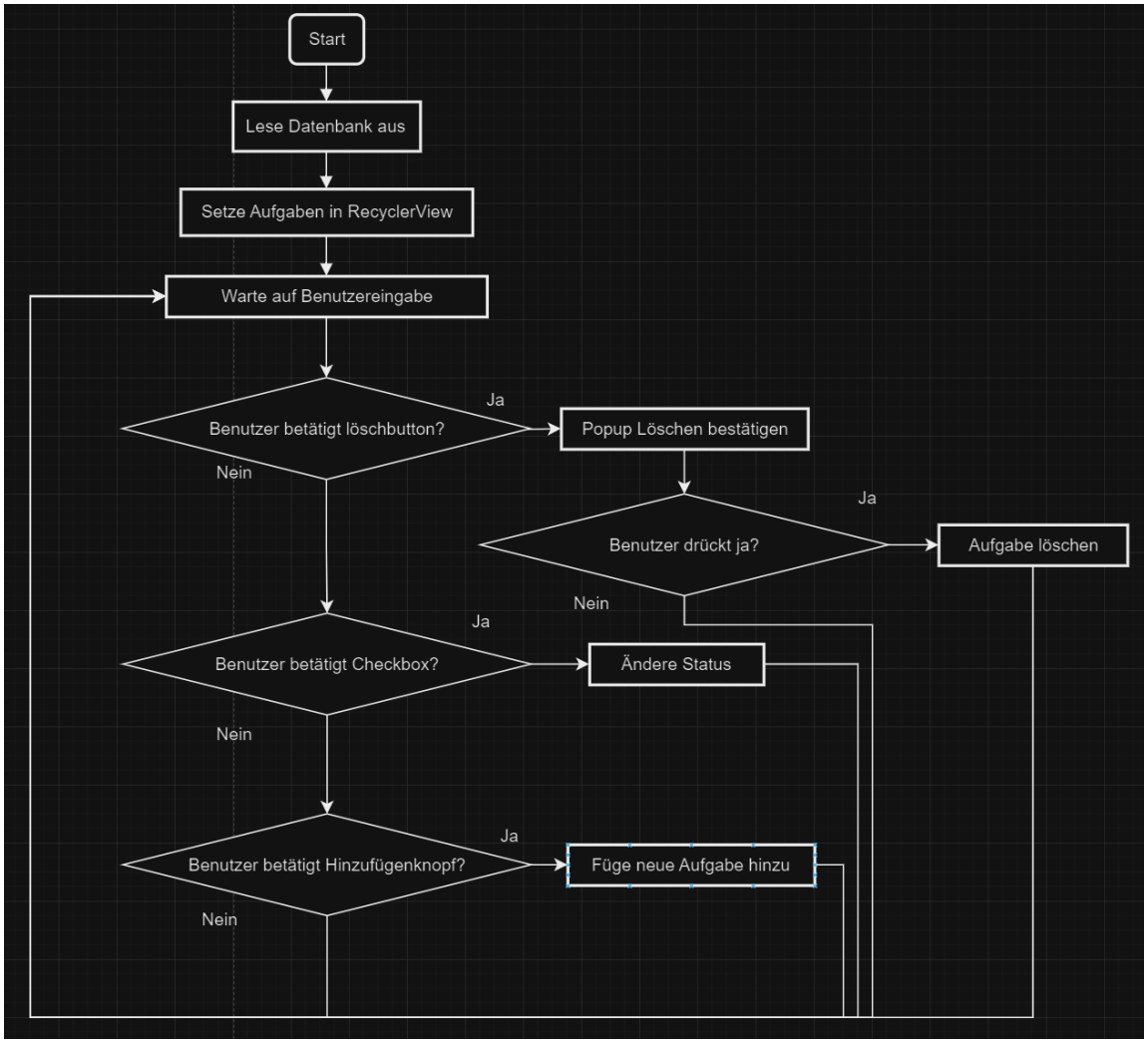
Lightmode

2.4 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 Android Apps erleichtert.

```
34    dependencies {
35        implementation("androidx.room:room-runtime:2.6.1")
36        annotationProcessor("androidx.room:room-compiler:2.6.1")
```

2.5 Ablaufdiagramm



3 Implementieren der 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.

3.1 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.

```
3  import androidx.room.ColumnInfo;
4  import androidx.room.Entity;
5  import androidx.room.PrimaryKey;
6
7  @Entity  73 usages
8  public class Aufgabe {
```

3.1.1 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.

```
10      @PrimaryKey(autoGenerate = true)  2 usages
11      private int id;
12      @ColumnInfo(name = "titel")  2 usages
13      private final String titel;
14      @ColumnInfo(name = "beschreibung")  2 usages
15      private final String beschreibung;
16      @ColumnInfo(name = "erledigt")  6 usages
17      private boolean erledigt;
```

3.1.2 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.

```
19      public Aufgabe(String titel, String beschreibung, Boolean erledigt) {
20          this.titel=titel;
21          this.beschreibung=beschreibung;
22          this.erledigt=erledigt;
23      }
```

3.1.3 Getter/Setter

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

```
25 >    public void changeErledigt() { erledigt=!erledigt; }
28
29 >    public int getId() { return id; }
32
33 >    public void setId(int id) { this.id = id; }
36
37 >    public String getTitel() { return titel; }
40
41 >    public String getBeschreibung() { return beschreibung; }
44
45 >    public boolean isErledigt() { return erledigt; }
```

3.2 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.

```
3   import androidx.room.Database;
4   import androidx.room.RoomDatabase;
5
6   @Database(entities = {Aufgabe.class}, version = 1)  9 usages  1 inheritor
7   public abstract class AppDatenbank extends RoomDatabase {
8       public abstract AufgabenDao aufgabenDao();  11 usages  1 implementation
9   }
10
```

3.2.1 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.

```
8       public abstract AufgabenDao aufgabenDao();
```

3.3 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.

```
3   import androidx.room.Dao;
4   import androidx.room.Delete;
5   import androidx.room.Insert;
6   import androidx.room.Query;
7   import androidx.room.Update;
8
9   import java.util.List;
10
11  @Dao  5 usages  1 implementation
12  public interface AufgabenDao {
```

3.3.1 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.

```
13  @Query("SELECT * FROM Aufgabe")  no usages  1 implementation
14  List<Aufgabe> getAllToDos();
15
16  @Query("SELECT * FROM Aufgabe WHERE erledigt = 1")  3 usages  1 implementation
17  List<Aufgabe> bekommeAlleErledigteAufgaben();
18
19  @Query("SELECT * FROM Aufgabe WHERE erledigt = 0")  3 usages  1 implementation
20  List<Aufgabe> bekommeAlleOffeneAufgaben();
21
22  @Insert  1 usage  1 implementation
23  void fuegeNeueAufgabeHinzu(Aufgabe aufgabe);
24
25  @Update  3 usages  1 implementation
26  void updateAufgabe(Aufgabe aufgabe);
27
28  @Delete  1 usage  1 implementation
29  void loescheToDo(Aufgabe aufgabe);
```

3.4 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]>.

3.4.1 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 offeneAufgaben (Aufgaben, die noch nicht abgeschlossen sind) und die andere Liste enthält alle ErledigteAufgaben (Aufgaben, die abgeschlossen sind). Der ErledigteAufgabenAdapter 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.

```
21  public class OffeneAufgabenAdapter extends RecyclerView.Adapter<OffeneAufgabenAdapter.OffeneAufgabenHolder> {
22      private final Context mainContext;  2 usages
23      private List<Aufgabe> offeneAufgabenListe;  10 usages
24      private List<Aufgabe> erledigteAufgabenListe;  7 usages
25      private ErledigteAufgabenAdapter erledigteAufgabenAdapter;  3 usages
26      private final AppDatenbank datenbank;  2 usages
27      private final Alarm alarm;  4 usages
28      private final TextView leereOffeneAufgabenNachrichtTextView;  3 usages
29      private final TextView leereErledigteAufgabenNachrichtTextView;  3 usages
30      private final RecyclerView offeneAufgabenRecyclerView;  3 usages
31      private final RecyclerView erledigteAufgabenRecyclerView;  3 usages
```

3.4.2 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.

```
34      public OffeneAufgabenAdapter(Context mainContext,  1 usage
35                                   Alarm alarm,
36                                   AppDatenbank datenbank,
37                                   TextView leereOffeneAufgabenNachrichtTextView,
38                                   TextView leereErledigteAufgabenNachrichtTextView,
39                                   RecyclerView offeneAufgabenRecyclerView,
40                                   RecyclerView erledigteAufgabenRecyclerView) {
41          this.mainContext = mainContext;
42          this.alarm = alarm;
43          this.datenbank = datenbank;
44          this.leereOffeneAufgabenNachrichtTextView = leereOffeneAufgabenNachrichtTextView;
45          this.leereErledigteAufgabenNachrichtTextView = leereErledigteAufgabenNachrichtTextView;
46          this.offeneAufgabenRecyclerView = offeneAufgabenRecyclerView;
47          this.erledigteAufgabenRecyclerView = erledigteAufgabenRecyclerView;
48      }
```

3.4.3 Setter

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

```
50      public void setErledigteAufgabenAdapter(ErledigteAufgabenAdapter erledigteAufgabenAdapter) {
51          this.erledigteAufgabenAdapter = erledigteAufgabenAdapter;
52      }
```

3.4.4 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.

```
62      @Override
63      public OffeneAufgabenHolder onCreateViewHolder(ViewGroup parent, int viewType) {
64          View view = LayoutInflater.from(this.mainContext).inflate(R.layout.offeneaufgaben_layout, parent,  attachToRoot: false);
65          return new OffeneAufgabenHolder(view);
66      }
```

3.4.5 Methode getItemCount

Diese überschriebene Methode gibt die Länge der Liste zurück.

```
68      @Override
69      public int getItemCount() { return this.offeneAufgabenListe.size(); }
72
```

3.4.6 Statische Klasse OffeneAufgabenHolder

Diese Klasse erbt die Klasse RecyclerView.ViewHolder. Diese Klasse spielt eine entscheidende Rolle für die effiziente Darstellung und Verwaltung der Daten innerhalb des RecyclerView. 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.

```
74      public static class OffeneAufgabenHolder extends RecyclerView.ViewHolder {  4 usages
75          private final TextView textTitel;  2 usages
76          private final CheckBox checkBox;  3 usages
77          private final ImageView loeschIcon;  2 usages
78          private final LinearLayout myLinearLayout;  2 usages
79
80
81          public OffeneAufgabenHolder(View itemView) {  1 usage
82              super(itemView);
83              this.myLinearLayout = itemView.findViewById(R.id.myLinearLayout);
84              this.textTitel = itemView.findViewById(R.id.textViewItem);
85              this.checkBox = itemView.findViewById(R.id.checkBox);
86              this.loeschIcon = itemView.findViewById(R.id.loeschIcon);
87
88          }
89
90  @ >     private void SetDetails(Aufgabe aufgabe) { this.textTitel.setText(aufgabe.getTitel()); }
93
94      }
```

3.4.7 Methode onBindViewHolder

In der Klasse wird Logik verwendet, um auf bestimmte Ereignisse zu reagieren. In diesem Projekt 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.

```java
    @Override
    public void onBindViewHolder(OffeneAufgabenHolder holder, int position) {
        Aufgabe gefundeneAufgabe = this.offeneAufgabenListe.get(position);
        holder.checkBox.setChecked(false);
        holder.SetDetails(gefundeneAufgabe);

        holder.checkBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                int adapterPosition = holder.getAdapterPosition();
                if (adapterPosition != RecyclerView.NO_POSITION) {
                    Aufgabe gecheckteAufgabe = offeneAufgabenListe.get(adapterPosition);
                    if (isChecked) {
                        offeneAufgabenListe.remove(gecheckteAufgabe);
                        gecheckteAufgabe.changeErledigt();
                        erledigteAufgabenListe.add(gecheckteAufgabe);
                        ExecutorService executor = Executors.newSingleThreadExecutor();
                        executor.execute(() -> {
                            datenbank.aufgabenDao().updateAufgabe(gecheckteAufgabe);
                        });
                        executor.shutdown();
                        offeneAufgabenListe.sort(Comparator.comparing(Aufgabe::getTitel));
                        erledigteAufgabenListe.sort(Comparator.comparing(Aufgabe::getTitel));

                        updateListen(offeneAufgabenListe, erledigteAufgabenListe);
                        erledigteAufgabenAdapter.updateListen(offeneAufgabenListe, erledigteAufgabenListe);
                        alarm.updateListen(offeneAufgabenListe, erledigteAufgabenListe);

                        if (offeneAufgabenListe.isEmpty()) {
                            leereOffeneAufgabenNachrichtTextView.setVisibility(View.VISIBLE);
                            offeneAufgabenRecyclerView.setVisibility(View.GONE);
                        } else {
                            leereOffeneAufgabenNachrichtTextView.setVisibility(View.GONE);
                            offeneAufgabenRecyclerView.setVisibility(View.VISIBLE);
                        }
                        if (erledigteAufgabenListe.isEmpty()) {
                            leereErledigteAufgabenNachrichtTextView.setVisibility(View.VISIBLE);
                            erledigteAufgabenRecyclerView.setVisibility(View.GONE);
                        } else {
                            leereErledigteAufgabenNachrichtTextView.setVisibility(View.GONE);
                            erledigteAufgabenRecyclerView.setVisibility(View.VISIBLE);
                        }
                        notifyItemRemoved(adapterPosition);
                        erledigteAufgabenAdapter.notifyDataSetChanged();
                    }
                }
            }
        });
        holder.loeschIcon.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {alarm.alertBestaetigeLoeschvorgang(gefundeneAufgabe);}
        });
        holder.myLinearLayout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {alarm.alertAufgabeDetails(gefundeneAufgabe);}
        });
    }
```

3.5 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]>.

3.5.1 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 offeneAufgaben (Aufgaben, die noch nicht abgeschlossen sind) und die zweite Liste enthält alle erledigteAufgaben (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.

```
3   > import ...
19
20  public class ErledigteAufgabenAdapter extends RecyclerView.Adapter<ErledigteAufgabenAdapter.ErledigteAufgabenHolder> {
21      private final Context mainContext;  2 usages
22      private List<Aufgabe> offeneAufgabenListe;  7 usages
23      private List<Aufgabe> erledigteAufgabenListe;  10 usages
24      private final AppDatenbank datenbank;  2 usages
25      private final OffeneAufgabenAdapter offeneAufgabenAdapter;  3 usages
26      private final Alarm alarm;  4 usages
27      private final TextView leereOffeneAufgabenNachrichtTextView;  3 usages
28      private final TextView leereErledigteAufgabenNachrichtTextView;  3 usages
29      private final RecyclerView offeneAufgabenRecyclerView;  3 usages
30      private final RecyclerView erledigteAufgabenRecyclerView;  3 usages
```

3.5.2 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.

```
33      public ErledigteAufgabenAdapter(Context mainContext,   1 usage
34                                      OffeneAufgabenAdapter offeneAufgabenAdapter,
35                                      Alarm alarm, AppDatenbank datenbank,
36                                      TextView leereOffeneAufgabenNachrichtTextView,
37                                      TextView leereErledigteAufgabenNachrichtTextView,
38                                      RecyclerView offeneAufgabenRecyclerView,
39                                      RecyclerView erledigteAufgabenRecyclerView) {
40          this.mainContext = mainContext;
41          this.offeneAufgabenAdapter = offeneAufgabenAdapter;
42          this.alarm = alarm;
43          this.datenbank = datenbank;
44          this.leereOffeneAufgabenNachrichtTextView = leereOffeneAufgabenNachrichtTextView;
45          this.leereErledigteAufgabenNachrichtTextView = leereErledigteAufgabenNachrichtTextView;
46          this.offeneAufgabenRecyclerView = offeneAufgabenRecyclerView;
47          this.erledigteAufgabenRecyclerView = erledigteAufgabenRecyclerView;
48      }
```

3.5.3 Methode updateListen

Diese Methode aktualisiert die Listen in der Klasse.

```
60
61      public void updateListen(List<Aufgabe> offeneAufgabenListe, List<Aufgabe> erledigteAufgabenListe) {
62          this.offeneAufgabenListe = offeneAufgabenListe;
63          this.erledigteAufgabenListe = erledigteAufgabenListe;
64      }
```

3.5.4 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.

```
50      @Override
51      public ErledigteAufgabenHolder onCreateViewHolder(ViewGroup parent, int viewType) {
52          View view = LayoutInflater.from(this.mainContext).inflate(R.layout.erledigteaufgaben_layout, parent, attachToRoot: false);
53          return new ErledigteAufgabenHolder(view);
54      }
```

3.5.5 Methode getItemCount

Diese überschriebene Methode gibt die Länge der Liste zurück.

```
56      @Override
57      public int getItemCount() { return this.erledigteAufgabenListe.size(); }
```

3.5.6 Statische Klasse ErledigteAufgabenHolder

Diese Klasse erbt die Klasse RecyclerView.ViewHolder. Diese Klasse spielt eine entscheidende Rolle für die effiziente Darstellung und Verwaltung der Daten innerhalb des RecyclerView. 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.

```
67      public static class ErledigteAufgabenHolder extends RecyclerView.ViewHolder {  4 usages
68          private final TextView textTitel;  2 usages
69          private final CheckBox checkBox;  3 usages
70          private final ImageView loeschIcon;  2 usages
71          private final LinearLayout myLinearLayout;  2 usages
72
73
74          public ErledigteAufgabenHolder(View itemView) {  1 usage
75              super(itemView);
76              this.myLinearLayout = itemView.findViewById(R.id.myLinearLayout);
77              this.textTitel = itemView.findViewById(R.id.textViewItem);
78              this.checkBox = itemView.findViewById(R.id.checkBox);
79              this.loeschIcon = itemView.findViewById(R.id.loeschIcon);
80
81          }
82
83  @>      void SetDetails(Aufgabe aufgabe) { this.textTitel.setText(aufgabe.getTitel()); }
86      }
```

3.5.7 Methode onBindViewHolder

In der Klasse wird Logik verwendet, um auf bestimmte Ereignisse zu reagieren. In diesem Projekt 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.

```
    public void onBindViewHolder(ErledigteAufgabenHolder holder, int position) {
        Aufgabe aufgabe = this.erledigteAufgabenListe.get(position);
        holder.checkBox.setChecked(true);
        holder.SetDetails(aufgabe);

        holder.checkBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                int adapterPosition = holder.getAdapterPosition();
                if (adapterPosition != RecyclerView.NO_POSITION) {
                    Aufgabe gecheckteAufgabe = erledigteAufgabenListe.get(adapterPosition);
                    if (!isChecked) {
                        erledigteAufgabenListe.remove(gecheckteAufgabe);
                        gecheckteAufgabe.changeErledigt();
                        offeneAufgabenListe.add(gecheckteAufgabe);
                        ExecutorService executor = Executors.newSingleThreadExecutor();
                        executor.execute(() -> {
                            datenbank.aufgabenDao().updateAufgabe(gecheckteAufgabe);
                        });
                        executor.shutdown();
                        offeneAufgabenListe.sort(Comparator.comparing(Aufgabe::getTitel));
                        erledigteAufgabenListe.sort(Comparator.comparing(Aufgabe::getTitel));

                        updateListen(offeneAufgabenListe, erledigteAufgabenListe);
                        offeneAufgabenAdapter.updateListen(offeneAufgabenListe, erledigteAufgabenListe);
                        alarm.updateListen(offeneAufgabenListe, erledigteAufgabenListe);

                        if (offeneAufgabenListe.isEmpty()) {
                            leereOffeneAufgabenNachrichtTextView.setVisibility(View.VISIBLE);
                            offeneAufgabenRecyclerView.setVisibility(View.GONE);
                        } else {
                            leereOffeneAufgabenNachrichtTextView.setVisibility(View.GONE);
                            offeneAufgabenRecyclerView.setVisibility(View.VISIBLE);
                        }
                        if (erledigteAufgabenListe.isEmpty()) {
                            leereErledigteAufgabenNachrichtTextView.setVisibility(View.VISIBLE);
                            erledigteAufgabenRecyclerView.setVisibility(View.GONE);
                        } else {
                            leereErledigteAufgabenNachrichtTextView.setVisibility(View.GONE);
                            erledigteAufgabenRecyclerView.setVisibility(View.VISIBLE);
                        }
                        notifyItemRemoved(adapterPosition);
                        offeneAufgabenAdapter.notifyDataSetChanged();
                    }

                }
            }
        });
        holder.loeschIcon.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { alarm.alertBestaetigeLoeschvorgang(aufgabe); }
        });
        holder.myLinearLayout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { alarm.alertAufgabeDetails(aufgabe); }
        });
    }

}
```

3.6 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.

3.6.1 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 noch 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 sowohl in der Anwendung als auch 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.

```
21  public class Alarm {  6 usages
22      private final Context mainContext;  7 usages
23      private List<Aufgabe> offeneAufgabenListe;  11 usages
24      private List<Aufgabe> erledigteAufgabenListe;  10 usages
25      private final AppDatenbank datenbank;  8 usages
26      private OffeneAufgabenAdapter offeneAufgabenAdapter;  3 usages
27      private ErledigteAufgabenAdapter erledigteAufgabenAdapter;  3 usages
28      private final Activity mainActivity;  4 usages
29      private final TextView leereOffeneAufgabenNachrichtTextView;  3 usages
30      private final TextView leereErledigteAufgabenNachrichtTextView;  3 usages
31      private final RecyclerView offeneAufgabenRecyclerView;  3 usages
32      private final RecyclerView erledigteAufgabenRecyclerView;  3 usages
```

3.6.2 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.

```
34      public Alarm(Activity mainActivity, Context mainContext, AppDatenbank datenbank,  1 usage
35                   TextView leereOffeneAufgabenNachrichtTextView,
36                   TextView leereErledigteAufgabenNachrichtTextView,
37                   RecyclerView offeneAufgabenRecyclerView,
38                   RecyclerView erledigteAufgabenRecyclerView) {
39          this.mainContext = mainContext;
40          this.mainActivity = mainActivity;
41          this.datenbank = datenbank;
42          this.leereOffeneAufgabenNachrichtTextView = leereOffeneAufgabenNachrichtTextView;
43          this.leereErledigteAufgabenNachrichtTextView = leereErledigteAufgabenNachrichtTextView;
44          this.offeneAufgabenRecyclerView = offeneAufgabenRecyclerView;
45          this.erledigteAufgabenRecyclerView = erledigteAufgabenRecyclerView;
46      }
```

3.6.3 Setter

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

```
48      public void setBeideAdapter(OffeneAufgabenAdapter offeneAufgabenAdapter, ErledigteAufgabenAdapter erledigteAufgabenAdapter) {
49          this.offeneAufgabenAdapter = offeneAufgabenAdapter;
50          this.erledigteAufgabenAdapter = erledigteAufgabenAdapter;
51      }
```

3.6.4 Methode updateListen

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

```
53      public void updateListen(List<Aufgabe> offeneAufgabenListe, List<Aufgabe> erledigteAufgabenListe) {
54          this.offeneAufgabenListe = offeneAufgabenListe;
55          this.erledigteAufgabenListe = erledigteAufgabenListe;
56      }
```

3.6.5 Methode alertAufgabeDetails

Diese Methode wurde erstellt, um die Details der Aufgabe anzuzeigen. Details wie ID, Titel, Beschreibung und 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.

```java
public void alertAufgabeDetails(Aufgabe uebergebeneAufgabe) {  2 usages
    AlertDialog.Builder builder = new AlertDialog.Builder(this.mainContext);
    View dialogView = LayoutInflater.from(this.mainContext).inflate(R.layout.aufgabendetails_layout,  root: null);
    TextView setID = dialogView.findViewById(R.id.setID);
    setID.setText(String.valueOf(uebergebeneAufgabe.getId()));
    TextView setTitel = dialogView.findViewById(R.id.setTitel);
    setTitel.setText(uebergebeneAufgabe.getTitel());
    TextView setBeschreibung = dialogView.findViewById(R.id.setBeschreibung);
    if (uebergebeneAufgabe.getBeschreibung().isEmpty()) {
        setBeschreibung.setText(R.string.keine_beschreibung_vorhanden);
    } else {
        setBeschreibung.setText(uebergebeneAufgabe.getBeschreibung());
    }
    CheckBox setCheckBox = dialogView.findViewById(R.id.setCheckBox);
    setCheckBox.setChecked(uebergebeneAufgabe.isErledigt());
    builder.setView(dialogView);
    setCheckBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            if (isChecked) {
                offeneAufgabenListe.remove(uebergebeneAufgabe);
                uebergebeneAufgabe.changeErledigt();
                erledigteAufgabenListe.add(uebergebeneAufgabe);
            } else {
                erledigteAufgabenListe.remove(uebergebeneAufgabe);
                uebergebeneAufgabe.changeErledigt();
                offeneAufgabenListe.add(uebergebeneAufgabe);
            }
            ExecutorService executor = Executors.newSingleThreadExecutor();
            executor.execute(() -> {
                datenbank.aufgabenDao().updateAufgabe(uebergebeneAufgabe);

                mainActivity.runOnUiThread(() -> {
                    updateUI();

                });
            });
            executor.shutdown();
        }
    });

    builder.setNeutralButton( text: "OK", new DialogInterface.OnClickListener() {
        @Override // android.content.DialogInterface.OnClickListener
        public void onClick(DialogInterface dialogInterface, int i) {

        }
    });

    builder.show();
}
```

3.6.6 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.

```
109 @    public void alertBestaetigeLoeschvorgang(Aufgabe uebergebeneAufgabe) {  2 usages
110          AlertDialog.Builder builder = new AlertDialog.Builder(this.mainContext);
111          builder.setMessage("Soll das ToDo '" + uebergebeneAufgabe.getTitel() + "' wirklich gelöscht werden?");
112          builder.setPositiveButton( text: "Ja", new DialogInterface.OnClickListener() {
113              @Override
114 @            public void onClick(DialogInterface dialogInterface, int i) {
115                  ExecutorService executor = Executors.newSingleThreadExecutor();
116                  executor.execute(() -> {
117                      datenbank.aufgabenDao().loescheToDo(uebergebeneAufgabe);
118                      offeneAufgabenListe = datenbank.aufgabenDao().bekommeAlleOffeneAufgaben();
119                      erledigteAufgabenListe = datenbank.aufgabenDao().bekommeAlleErledigteAufgaben();
120
121                      mainActivity.runOnUiThread(() -> {
122                          updateUI();
123                      });
124
125                  });
126                  executor.shutdown();
127              }
128          });
129          builder.setNegativeButton( text: "Nein", new DialogInterface.OnClickListener() {
130              @Override // android.content.DialogInterface.OnClickListener
131 @            public void onClick(DialogInterface dialogInterface, int i) {
132
133              }
134          });
135          builder.show();
136      }
```

3.6.7 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 erledigt immer auf false gesetzt.

```
138     public void alertNeueAufgabeErstellen() {  1 usage
139         AlertDialog.Builder builder = new AlertDialog.Builder(this.mainContext);
140         View dialogView = LayoutInflater.from(this.mainContext).inflate(R.layout.dialog_neue_todo,  root: null);
141         EditText inputTitel = dialogView.findViewById(R.id.inputTitel);
142         EditText inputBeschreibung = dialogView.findViewById(R.id.inputBeschreibung);
143         builder.setView(dialogView);
144         builder.setPositiveButton( text: "Hinzufügen", new DialogInterface.OnClickListener() {
145             @Override // android.content.DialogInterface.OnClickListener
146             public void onClick(DialogInterface dialogInterface, int i) {
147                 if (inputTitel.getText().toString().isEmpty()) {
148                     alertEmptyTitel();
149                 } else {
150                     Aufgabe neueAufgabe = new Aufgabe(inputTitel.getText().toString(), inputBeschreibung.getText().toString(),  erledigt: false);
151 
152                     ExecutorService executor = Executors.newSingleThreadExecutor();
153                     executor.execute(() -> {
154                         datenbank.aufgabenDao().fuegeNeueAufgabeHinzu(neueAufgabe);
155                         offeneAufgabenListe.add(neueAufgabe);
156 
157                         offeneAufgabenListe = datenbank.aufgabenDao().bekommeAlleOffeneAufgaben();
158                         erledigteAufgabenListe = datenbank.aufgabenDao().bekommeAlleErledigteAufgaben();
159 
160                         mainActivity.runOnUiThread(() -> {
161                             updateUI();
162                         });
163 
164                     });
165                     executor.shutdown();
166 
167                 }
168             }
169         });
170         builder.setNegativeButton( text: "Beenden", new DialogInterface.OnClickListener() {
171             @Override // android.content.DialogInterface.OnClickListener
172             public void onClick(DialogInterface dialogInterface, int i) {
173 
174             }
175         });
176         builder.show();
177     }
```

3.6.8 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.

```
178     public void alertEmptyTitel() {  1 usage
179         AlertDialog.Builder builder = new AlertDialog.Builder(this.mainContext);
180         builder.setMessage("Aufgabe kann nicht erstellt werden. Titel darf nicht leer bleiben");
181         builder.setNeutralButton( text: "OK", new DialogInterface.OnClickListener() {
182             @Override
183             public void onClick(DialogInterface dialog, int which) {
184 
185             }
186         });
187         builder.show();
188     }
```

3.6.9 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.

```
191     private void updateUI() {  3 usages
192         offeneAufgabenListe.sort(Comparator.comparing(Aufgabe::getTitel));
193         erledigteAufgabenListe.sort(Comparator.comparing(Aufgabe::getTitel));
194
195         offeneAufgabenAdapter.updateListen(offeneAufgabenListe, erledigteAufgabenListe);
196         erledigteAufgabenAdapter.updateListen(offeneAufgabenListe, erledigteAufgabenListe);
197         updateListen(offeneAufgabenListe, erledigteAufgabenListe);
198
199         if (offeneAufgabenListe.isEmpty()) {
200             leereOffeneAufgabenNachrichtTextView.setVisibility(View.VISIBLE);
201             offeneAufgabenRecyclerView.setVisibility(View.GONE);
202         } else {
203             leereOffeneAufgabenNachrichtTextView.setVisibility(View.GONE);
204             offeneAufgabenRecyclerView.setVisibility(View.VISIBLE);
205         }
206         if (erledigteAufgabenListe.isEmpty()) {
207             leereErledigteAufgabenNachrichtTextView.setVisibility(View.VISIBLE);
208             erledigteAufgabenRecyclerView.setVisibility(View.GONE);
209         } else {
210             leereErledigteAufgabenNachrichtTextView.setVisibility(View.GONE);
211             erledigteAufgabenRecyclerView.setVisibility(View.VISIBLE);
212         }
213         offeneAufgabenAdapter.notifyDataSetChanged();
214         erledigteAufgabenAdapter.notifyDataSetChanged();
215     }
```

3.7 Implementieren der Klasse MainActivity

Diese Klasse wird immer automatisch generiert, wenn ein neues Projekt erstellt wird. In dieser Klasse wird die gesamte Anwendung beim Start gestartet.

3.7.1 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.

```
20  public class MainActivity extends AppCompatActivity {
21
22      List<Aufgabe> offeneAufgabenListe = new ArrayList<>();  7 usages
23      List<Aufgabe> erledigteAufgabenListe = new ArrayList<>();  7 usages
24
25      private Alarm alarm;  6 usages
26      private OffeneAufgabenAdapter offeneAufgabenAdapter;  8 usages
27      private ErledigteAufgabenAdapter erledigteAufgabenAdapter;  7 usages
28
29      private TextView leereOffeneAufgabenNachrichtTextView;  6 usages
30      private TextView leereErledigteAufgabenNachrichtTextView;  6 usages
31      private RecyclerView offeneAufgabenRecyclerView;  8 usages
32      private RecyclerView erledigteAufgabenRecyclerView;  8 usages
33
34      private AppDatenbank datenbank;  6 usages
35      private final ExecutorService executor = Executors.newSingleThreadExecutor();
```

3.7.2 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.

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

    FloatingActionButton neueAufgabeKnopf = findViewById(R.id.addAufgabeBtn);

    leereOffeneAufgabenNachrichtTextView = findViewById(R.id.empty_text_view1);

    leereErledigteAufgabenNachrichtTextView = findViewById(R.id.empty_text_view2);

    offeneAufgabenRecyclerView = findViewById(R.id.myRecyclerView1);

    erledigteAufgabenRecyclerView = findViewById(R.id.myRecyclerView2);

    datenbank = Room.databaseBuilder(getApplicationContext(), AppDatenbank.class, name: "aufgaben.db").build();

    alarm = new Alarm( mainActivity: this, mainContext: this, datenbank, leereOffeneAufgabenNachrichtTextView,
            leereErledigteAufgabenNachrichtTextView, offeneAufgabenRecyclerView, erledigteAufgabenRecyclerView);

    offeneAufgabenAdapter  = new OffeneAufgabenAdapter( mainContext: this, alarm, datenbank,
            leereOffeneAufgabenNachrichtTextView, leereErledigteAufgabenNachrichtTextView,
            offeneAufgabenRecyclerView, erledigteAufgabenRecyclerView);

    erledigteAufgabenAdapter  = new ErledigteAufgabenAdapter( mainContext: this, offeneAufgabenAdapter, alarm,
            datenbank, leereOffeneAufgabenNachrichtTextView, leereErledigteAufgabenNachrichtTextView,
            offeneAufgabenRecyclerView, erledigteAufgabenRecyclerView);

    offeneAufgabenAdapter.setErledigteAufgabenAdapter(erledigteAufgabenAdapter);

    offeneAufgabenRecyclerView.setLayoutManager(new LinearLayoutManager(getApplicationContext()));
    offeneAufgabenRecyclerView.setAdapter(this.offeneAufgabenAdapter);

    erledigteAufgabenRecyclerView.setLayoutManager(new LinearLayoutManager(getApplicationContext()));
    erledigteAufgabenRecyclerView.setAdapter(this.erledigteAufgabenAdapter);

    alarm.setBeideAdapter(offeneAufgabenAdapter, erledigteAufgabenAdapter);

    ladeAufgabenUndUpdateUI();
    neueAufgabeKnopf.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            alarm.alertNeueAufgabeErstellen();
            offeneAufgabenAdapter.notifyDataSetChanged();
            erledigteAufgabenAdapter.notifyDataSetChanged();
        }
    });

}
```

3.7.3 Methode ladeAufgabenUndUpdateUI

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

```
88      private void ladeAufgabenUndUpdateUI() {  1 usage
89          executor.execute(() -> {
90              leseDatenbankUndAkualisierListen();
91
92              runOnUiThread(() -> {
93                  aktualisiereViewSichtbarkeit();
94                  offeneAufgabenAdapter.notifyDataSetChanged();
95                  erledigteAufgabenAdapter.notifyDataSetChanged();
96              });
97          });
98          executor.shutdown();
99      }
```

3.7.4 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.

```
101     private void leseDatenbankUndAkualisierListen() {  1 usage
102         offeneAufgabenListe = datenbank.aufgabenDao().bekommeAlleOffeneAufgaben();
103         erledigteAufgabenListe = datenbank.aufgabenDao().bekommeAlleErledigteAufgaben();
104
105         offeneAufgabenListe.sort(Comparator.comparing(Aufgabe::getTitel));
106         erledigteAufgabenListe.sort(Comparator.comparing(Aufgabe::getTitel));
107
108         offeneAufgabenAdapter.updateListen(offeneAufgabenListe, erledigteAufgabenListe);
109         erledigteAufgabenAdapter.updateListen(offeneAufgabenListe, erledigteAufgabenListe);
110         alarm.updateListen(offeneAufgabenListe, erledigteAufgabenListe);
111     }
```

3.7.5 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.

```
113     private void aktualisiereViewSichtbarkeit() {  1 usage
114         if (offeneAufgabenListe.isEmpty()) {
115             leereOffeneAufgabenNachrichtTextView.setVisibility(View.VISIBLE);
116             offeneAufgabenRecyclerView.setVisibility(View.GONE);
117         } else {
118             leereOffeneAufgabenNachrichtTextView.setVisibility(View.GONE);
119             offeneAufgabenRecyclerView.setVisibility(View.VISIBLE);
120         }
121         if (erledigteAufgabenListe.isEmpty()) {
122             leereErledigteAufgabenNachrichtTextView.setVisibility(View.VISIBLE);
123             erledigteAufgabenRecyclerView.setVisibility(View.GONE);
124         } else {
125             leereErledigteAufgabenNachrichtTextView.setVisibility(View.GONE);
126             erledigteAufgabenRecyclerView.setVisibility(View.VISIBLE);
127         }
128     }
```

3.8 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.

3.8.1 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.

3.8.2 Layouts

Das Layout ist frei wählbar. Die beiden RecyclerViews, in denen die Aufgaben angezeigt werden, sind die wichtigsten Elemente dieses Projekts. Der Autor hat sich jedoch die Mühe gemacht, diese Anwendung etwas schöner aussehen zu lassen, indem er die Items auf der RecyclerView abgerundet und die Aktionen mit kleinen Bildern gekennzeichnet hat. Außerdem wurde ein neues Attribut (itemBackgroundColor) vom Format Color erstellt, mit dem die Hintergrundfarbe des Items mit Hilfe der Themes geändert werden kann.

3.8.3 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.

4 Nützliche Links

Dynamische Listen mit RecyclerView erstellen | Views | Android Developers

RecyclerView.Adapter | Android Developers

RecyclerView.ViewHolder | Android Developers

Daten mit Room in einer lokalen Datenbank speichern | Android Developers

Dialogfelder | Views | Android Developers

Stile und Designs | Views | Android Developers