PHOTO JSL

Dokumentation zum Android Projekt

Jendrik Jordan, Simon Oswald und Louisa Pabst
[E-Mail-Adresse]

Inhalt

1. „Photo JSL“-Übersicht

Name	Photo JSL
Version	0.1
Programmiersprache	Java
Entwicklungsumgebung	Android Studio 1.3.2
Entwickler	Jendrik Jordan, Simon Oswald und Louisa Pabst
Android Version	Lollipop 5.1
Testgerät	Nexus 5 mit Android 6.0
Entwickelt für	Android 5.0 und höher

Die Android entwickelte App „Photo JSL“ bietet dem Nutzer die Bildaufnahme und -verwaltung. Der Nutzer hat die Möglichkeit in der Fotobibliothek der App seine selbstaufgenommenen Fotos zu verwalten und zusätzlich von außerhalb Bilder in diese zu importieren. Alle Bilder werden übersichtlich für den Nutzer mit einem Thumbnail-Bild, dem Dateiname und dem Datum und Zeit der Bildaufnahme auf dem Startfenster dargestellt. Zusätzlich bietet die App die Funktion Bilder direkt über Email an Bekannte und Freunde zu verschicken. Das aktuelle Produkt ist zurzeit noch in der alpha-Phase (0.1), in der alle grundlegenden angeforderten Funktionen bereits implementiert wurden. Um die App markttauglich zu gestalten wäre eine Möglichkeit diese mit beliebten Bildbearbeitungen und Filtern weiterzuentwickeln. Durch die Anforderung des Kunden externe Bilder zu importieren, aber auch Bilder aus der App zu exportieren, könnte der Nutzer externe Bilder zum Bearbeiten importieren und die für spätere Zwecke wieder zu exportieren.

Im Folgenden werden die Implementierungen der einzelnen Klassen beschrieben. Die Anforderungen, die wir vom Kunden (Dozent, Herr Sommer) erhalten werden, sind Klassen übergreifend. Deswegen werden wir, um Wiederholungen in den Beschreibungen zu vermeiden, jede Klasse für sich beschreiben und Bezug auf die Anforderungen machen. Bei jedem Klassenkapitel wird zuerst eine Kurzbeschreibung gemacht und danach wird näher auf die einzelnen Methoden der jeweiligen Klasse eingegangen. Es werden nur für die Beschreibung relevante Codeschnipsel in diesem Dokument aufgeführt, weil der vollständige Code dem Kunden zur Verfügung steht.

2. Klasse – „MainMenu“

1. „Main Menu“ – UML Diagramm

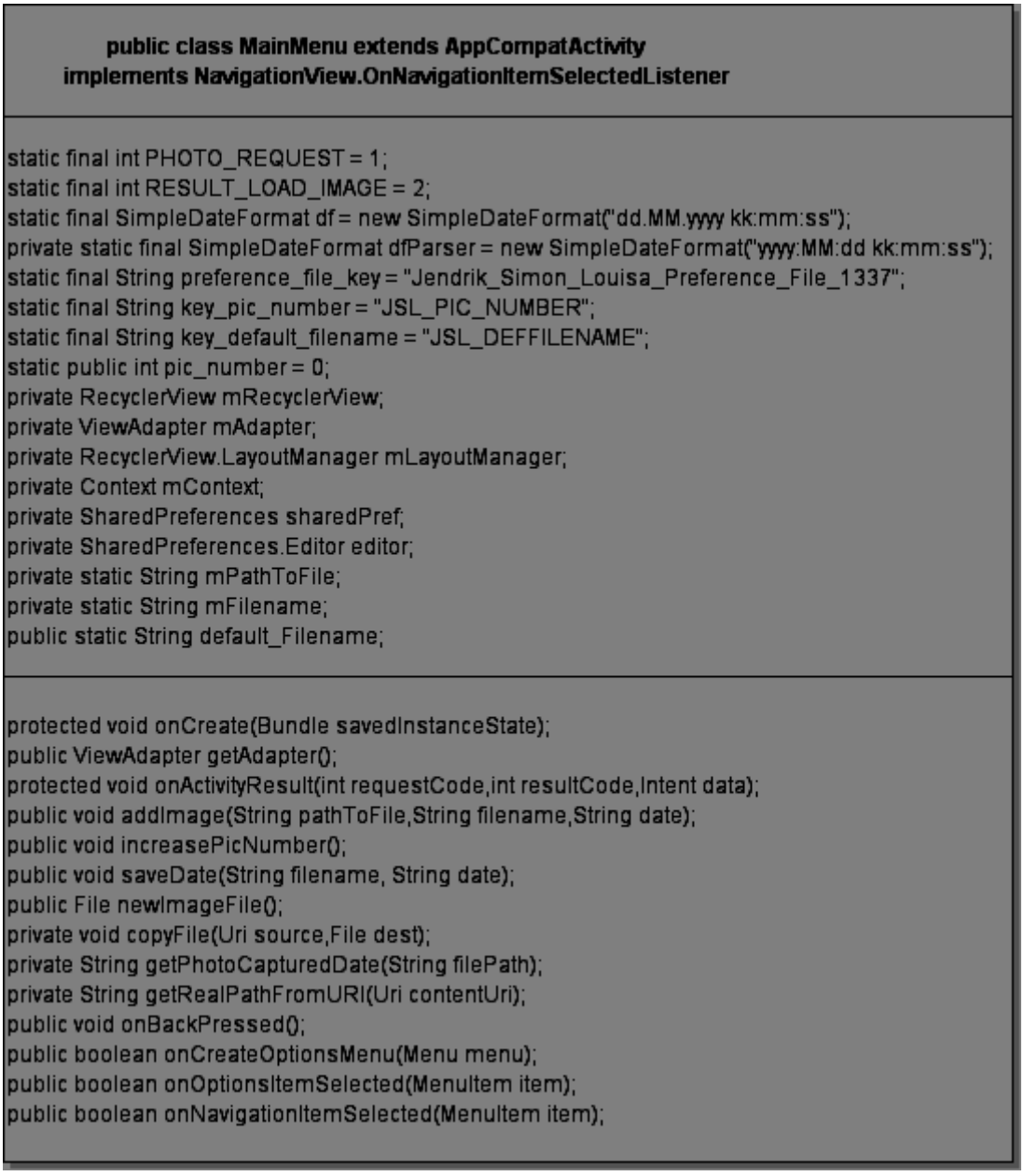

Abbildung 1 - "MainMenu" UML Diagramm

2. Kurzbeschreibung

Die Klasse „**MainMenu**“ repräsentiert das Hauptfenster der „App“. Sie stellt die Grund-Funktionen zum Darstellen der Bilder und der Navigation in weitere Fenster zur Verfügung. Sie erbt von „AppCompatActivity“ und implementiert das Interface „NavigationView.OnNavigationItemSelectedListener“.

3. Methoden

Die Methode „**onCreate**“ aus der Super-Klasse „AppCompatActivity“ wird überschrieben und dient zum Initialisieren der Klasse. Zuerst wird die Methode der Super-Klasse aufgerufen bevor mit der

eigenen Implementierung fortgefahren wird. Mit dem Aufruf der Funktion „setContentView“ wird als nächstes die Oberfläche spezifiziert, die in der XML-Datei „R.layout.activity_main_menu“ beschrieben ist. Die „Toolbar“ wird über die Funktion „findViewById“ und den Parameterwert „R.id.toolbar“ gefunden und mithilfe von „setSupportActionBar“ gesetzt. Als nächstes werden die Member-Variablen initialisiert. Zur Darstellung wird ein „Recycler-View“ mit einem „LinearLayoutManager“ benutzt. Gespeicherte Daten werden über die Funktion „getSharedPreferences“ des Kontextes abgerufen mithilfe des „preference_file_key“, welcher den Wert "Jendrik_Simon_Louisa_Preference_File_1337" hat. Die „Member-Variablen“ „pic_number“ und default_Filename werden aus den „sharedPref“ ausgelesen über key_pic_number="JSL_PIC_NUMBER" und key_default_filename= "JSL_DEFFILENAME". Im nachfolgenden Abschnitt wird das Verhalten des „Buttons“ spezifiziert, mit dem ein neues Foto gemacht werden kann. Dieser „FloatingActionButton“ wird wieder über „findViewById“ geholt und anschliessend mit einem „OnClickListener“ versehen. Die Funktion „onClick“ erstellt zuerst einen neuen „Intent“ mit „MediaStore.ACTION_IMAGE_CAPTURE“. Als nächstes wird die Ausgabedatei mithilfe von „newImageFile“ erstellt und die „Member-Variablen“ „mFilename“ und „mPathToFile“ entsprechend aktualisiert. Bevor der „Intend“ gestartet wird über „startActivityForResult“ und dem Wert „PHOTO_REQUEST“, wird EXTRA_OUTPUT noch mit der URI der Ausgabedatei versehen.

Die Methode „**getAdapter**“ fungiert als reiner „Getter“.

Die Methode „**onActivityResult**“ behandelt zwei Fälle bei erfolgreichem Abschließen der Aktivität:

1. Ein Foto wurde frisch aufgenommen (PHOTO_REQUEST). Die Methode „addImage“ wird aufgerufen.
2. Ein Foto wurde geladen (RESULT_LOAD_IMAGE). In diesem Fall wird das selektierte Bild über den mitgegebenen „Intend“ und die Methode „copyFile“ kopiert in ein neues „ImageFile“, welches mit „newImageFile“ erstellt wurde. Dem neuen Bild wird das Datum mitgegeben, falls es noch keines hatte. Anschließend wird das Bild hinzugefügt über „addImage“.

Die Methode „**addImage**“ fügt der Liste einen Eintrag mit dem Bild als „Thumbnail“ hinzu. Dazu wird die Methode „loadThumbFromFile“ aus dem Model genutzt mit dem übergebenen Pfad. Dem Adapter wird über „addData“ der neue „ListEntry“ hinzugefügt und zuvor das Datum abgespeichert mithilfe von „saveDate“. Letztlich muss noch „increasePicNumber“ aufgerufen werden, was im Folgenden beschrieben ist.

Die Methode „**increasePicNumber**“ inkrementiert die Member-Variable pic_number und setzt den Wert gleichzeitig über den „editor“ in „sharedPref“.

Die Methode „**saveDate**“ mappt über den „editor“ in „sharedPref“ die übergebenen Werte von „filename“ und „date“.

Die Methode „**newImageFile**“ erstellt eine neue Datei mit dem Namen, der sich zusammensetzt aus „default_Filename“ und der aktuellen Bildnummer („pic_number“).

Die Methode „**copyFile**“ kopiert eine Datei von einer gegebenen „URI“ in eine andere, geben durch die Datei an sich. Dazu wird ein „BufferedInputStream“ erstellt auf die Quelldatei und ein „BufferedOutputStream“ auf die Zieldatei. Anschließend wird ein „Buffer“ erzeugt mit der Größe von 1024 „Bytes“ und angefangen, über den Befehl „in.read(buffer)“ innherhalb einer „do-while“-Schleife, die Quelldatei auszulesen. Im „Body“ der Schleife wird das Gelesene in die Zieldatei über „out.write(buffer)“ geschrieben. Umgeben ist die Prozedur von einem „Try-Catch-Block“ damit eventuelle „Exceptions“ abgefangen werden. Unter „finally“ werden die „In“- und „OutputStreams“ geschlossen, falls sie geöffnet wurden.

Die Methode „**getPhotoCapturedDate**“ liest aus einerBilddatei das Erstelldatum aus mithilfe des „dfParser“ und einem „ExifInterface“. Dabei ist das „SimpleDateFormat“ auf „dd.MM.yyyy kk:mm:ss“ spezifiziert.

Die Methode „**getPathFromURI**“ liefert den tatsächlichen Pfad anhand einer gegebenen URI. Mithilfe eines „CursorLoader“und „Cursor“ wird innerhalb von „Mediastore.Images.Media.DATA“ an den Spalten-Index gesprungen nachdem der „Cursor“ an den Anfang geschoben wurde mit „cursor.moveToFirst“. Anschließend liefert „cursor.getString“ mit dem übergebenen Spalten-Index den Pfad in Form eines „String“, welcher zurückgegeben wird.

Die Methode „**onCreateOptionsMenu**“ setzt das Optionen-Menü, welches in der xml „R.menu.main_menu“ beschrieben ist, mithilfe des Funktionsaufrufs „inflate“ des „MenuInflater“.

Die Methode „**onOptionsItemSelected**“ kümmert sich um Aktionen im Optionen-Menü. Über das übergebene „Item“ wird die „ID“ ermittelt anhand derer das weitere Vorgehen entschieden wird:

1. „R.id.action.gallery“: Ein Bild aus der „Gallery“ soll importiert werden. Dazu wird ein neuer „Intent“ erstellt mit dem Wert „ACTION_PICK“ und aktiviert mit „startActivityForResult“ mit dem Wert „RESULT_LOAD_IMAGE“.
2. „R.id.action_settings“: Das Einstellungen-Menü soll aufgerufen werden. Dazu wird ein neuer „Intent“ erstellt mit dem Wert „mContext“ und aktiviert mit „startActivity“.

3. Klasse – „DetailView“

1. „DetailView“ – UML Diagramm

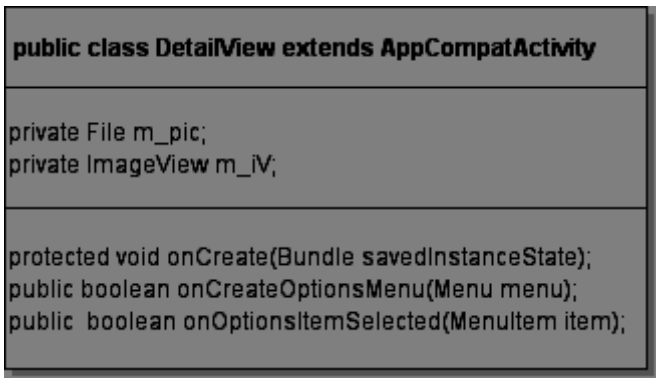

Abbildung 2 - "DetailView" UML Diagramm

2. Kurzbeschreibung

Die Klasse „**DetailView**“ erbt von „AppCompatActivity“ und beschreibt das Fenster zur Detailansicht eines Bildes. Möglich sind verschiedene Aktionen auf das Bild, wie das Löschen, Speichern oder Senden des Bildes. Das anzuzeigende Bild und der „ImageView“ werden als „Member“-Variablen „m_pic“ und „m_iV“ gehalten.

3. Methoden

Die Methode „**onCreate**“ wird beim Laden des „DetailView“ aufgerufen und initialisiert die „Member“-Variablen und setzt das Layout mit der XML-Datei „R.layout.activity_detail_view“. Beim Initialisieren der „Member“-Variablen wird der Dateiname aus dem aktuellen „Intent“ ausgelesen und der „ImageView“ mit einer entsprechend neu erstellten Datei versorgt.

Die Methode „**onCreateOptionsMenu**“ definiert das Menu über den Methodenaufruf von „getMenuInflater().inflate“ und den Parameterwert „R.menu.detail_menu“.

Die Methode „**onOptionsItemSelected**“ kümmert sich um Interaktionen mit dem Menü. Anhand der „ID“ aus dem übergebenen „MenuItem“ wird die Aktion erkannt.

1. „R.id.action_delete“: Das Bild soll gelöscht werden.
2. „R.id.action_send“: Das Bild soll per Email versendet werden. Dazu wird einen neuer „Intent“ mit dem Wert „ACTION_SEND“ erstellt, und dessen „Extra“ mithilfe der Funktion „putExtra“ gesetzt. „EXTRA_SUBJECT“ bekommt den Namen des Bildes, als Text wird „powered by JLS Software“ gesetzt und das Bild wird über „EXTRA_STREAM“ und der zuvor ermittelten URI mitgegeben. Der Typ der Bild-Datei wird noch als „png“ angegeben und schließlich wird die Aktivität gestartet mithilfe von „startActivity“ und der Aufforderung „Choose an application to share your picture“als Wert.
3. „R.id.action_save“: Das Bild soll in die „Gallery“ abgespeichert werden.

4. Klasse – „Model“

1. „Model“ – UML Diagramm

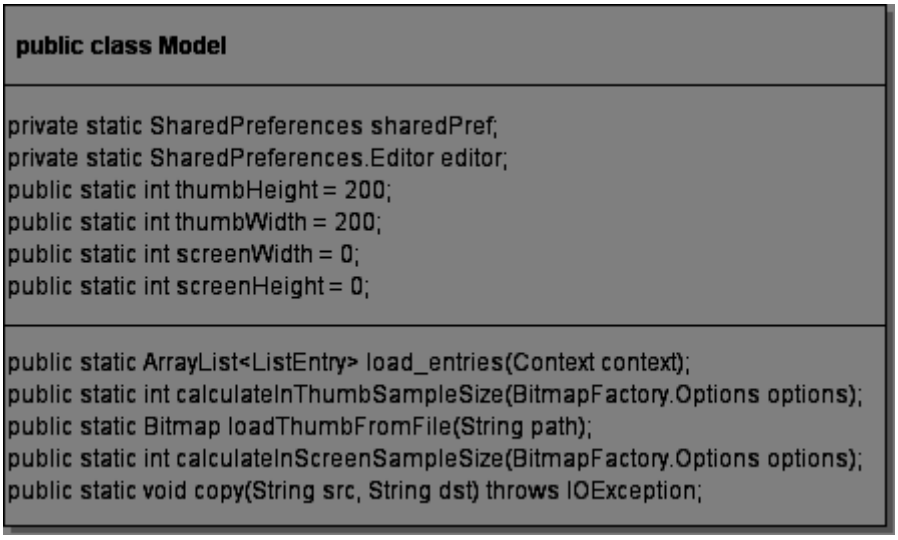

Abbildung 3 - "Model" UML Diagramm

2. Kurzbeschreibung

Die Klasse „**Model**“ kapselt alle Funktionen, die auf unterster technischer Ebene arbeiten. Sie bietet Methoden zum Arbeiten mit Thumbnails, dem Berechnen einer angepassten Bildgröße und dem Laden von Einträgen. Als Attribute enthält sie die „sharedPref“ mit einem „Editor“ darauf. Außerdem sind Höhe und Breite von „Thumbnails“ abgespeichert als „thumbHeight“ und „thumbWidth“.

3. Methoden

Die Methode „**load_entries**“ lädt alle Dateien aus dem Verzeichnis und fügt sie als „ListEntry“ einer „ArrayList“ hinzu, welche später zurückgegeben wird. Dazu wird diese zuerst erstellt. Anschliessend werden über „folder.listFiles()“ alle Dateien in das „Array“ „files“ geladen. Bevor die Dateien einzeln bearbeitet werden, werden noch die Member-Variablen „sharedPref“ und „editor“ gesetzt mithilfe des „preference_file_key“, der in „MainMenu“ definiert ist. Innerhalb der „For“-Schleife wird für jede Datei der absolute Pfad ausgelsen und über die Funktion „loadThumbFromFile“ eine neue Bitmap erstellt, welche danach als neuer „ListEntry“ den „listEntris“ mit „add“ hinzugefügt wird.

Die Methode „**calculateInThumbSampleSize**“ wurde aus der Google-Dokumentation übernommen und ist deswegen nicht weiter erklärt.

Die Methode „**loadThumbFromFile**“ lädt zu einem gegebenen Pfad die entsprechende Bitmap. Dazu wird die Variable „options“ vom Typ „BitmapFactory.Options“ dazu verwendet, die „inSampleSize“ und „inJustDecodeBounds“ zu spezifizieren. Vor dem Berechnen der „inSampleSize“ mithilfe der Funktion „calculateInThumbSampleSize“ gilt „inJustDecodeBounds=true“, während dannach „inJustDecodeBounds=false“ gilt. Anschließend wird die Bitmap dekodiert mit

„BitmapFactory.decodeFile“. Schließlich wird die Funktion „Bitmap.createBitmap“ dazu genutzt, um die Bitmap zu kreieren. Die Koordinaten werden durch die Berechnung „b.getWidth()/2 – thumbWidth/2“ (für „Height“ analog) ermittelt, wodurch die Koordinaten zentriert werden. Die Breite und Höhe wird einfach über „thumbWidth“ und „thumbHeight“ mitgegeben. Letztlich wird diese Bitmap zurückgegeben.

Die Methode „**calculateInScreenSampleSize**“ liefert zu gegebenen Bildmaßen die, an die Bildschirmmaßen angepassten, Maße für das Bild. Dazu werden zuerst die Höhe und die Breite des Bildes aus „options“ ausgelesen und in „height“ und „width“ gespeichert. Der Initialwert von „inSampleSize“ beträgt „1“. Anschliessen wird geprüft, ob „width“ oder „height“ größer sind als die Bildschirmmaße. Wenn dem so ist, wird das Bild um den Faktor 2 verkleinert. Falls diese Abmessungen noch immer zu groß sind, so wird innerhalb einer „while“-Schleife so lange inSampleSize mit „2“ multipliziert, bis „halfHeight/inSampleSize<=screenHeight“ und „halfWidth/inSampleSize<=screenWidth“ gilt. Die berechnete „inSampleSize“ wird zurückgegeben.

Die Methode „**copy**“ kopiert den Inhalt aus einer Datei in eine andere. Zuerst werden zwei „Streams“ geöffnet zum Schreiben und Lesen der Dateien. Bevor nun kopiert wird, wird ein „Buffer“ erzeugt mit der Größe von „1024“ „Bytes“. Das Kopieren passiert innerhalb einer „while“-Schleife, in der so lange mit „in.read“ „Bytes“ aus der Quelle gelesen werden und anschließend mit „out.write“ geschrieben werden, bis „in.read“ keine weiteren „Bytes“ mehr liefert. Die geöffneten Streams werden schließlich geschlossen mit „close“.

5. Klasse – „ListEntry“

1. „ListEntry“ – UML Diagramm

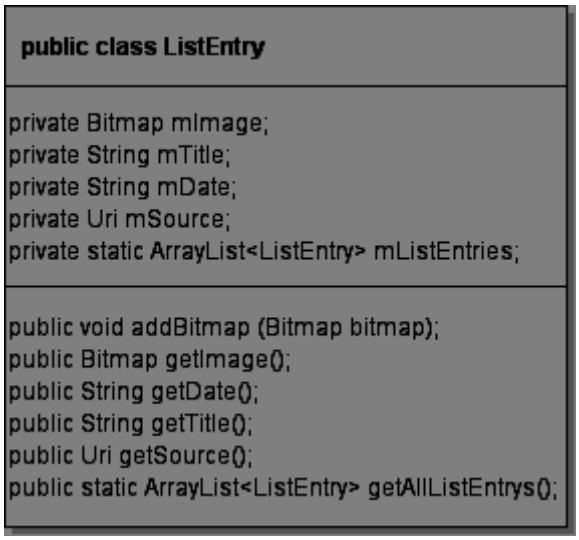

Abbildung 4 - "ListEntry UML Diagramm

2. Kurzbeschreibung

Die Klasse „**ListEntry**“ gibt die Datenstruktur an mit deren Hilfe ein Bild gespeichert wird. Zu jedem Bild werden das dazugehörige Datum mit Uhrzeit und der Dateiname, also Titel, abgespeichert. Desweiteren sind die getter-Methoden der Bitmapdatei, des Strings „Datum“ und des Strings „Titel“ in dieser Klasse.

3. Methoden und Constructor

Die Methoden „**getImage**()“, „**getDate**()“, „**getTitle**()“, sowie die Methode „**getAllListEntrys**(), geben die jeweiligen Attribute zurück. „**GetImage**()“ gibt ein Bitmap zurück, die Methoden „**getDate**()“ und „**getTitle**()“ geben einen String zurück. Wenn nicht ein einzelner Datentyp des Bilds gebraucht wird, sondern alle drei zusammen, wird die „Getter“-Methode „**getAllListEntrys**()“ genutzt. Diese gibt in einem Array die drei Datentypen in folgender Reihenfolge aus: Bitmap, Titel und Datum.

Des Weiteren wird ein Parameter Konstruktor „**ListEntry**“ angelegt. Dieser übergibt den private Parametern „mImage“, „mTitle“ und „mDate“ die jeweils übergegebenen Parametern. Zum Schluss werden diese Werte zu der Arrayliste „mListEntries“ hinzugefügt.

6. Klasse – „ViewAdapter“

1. „ViewAdapter“ – UML Diagramm

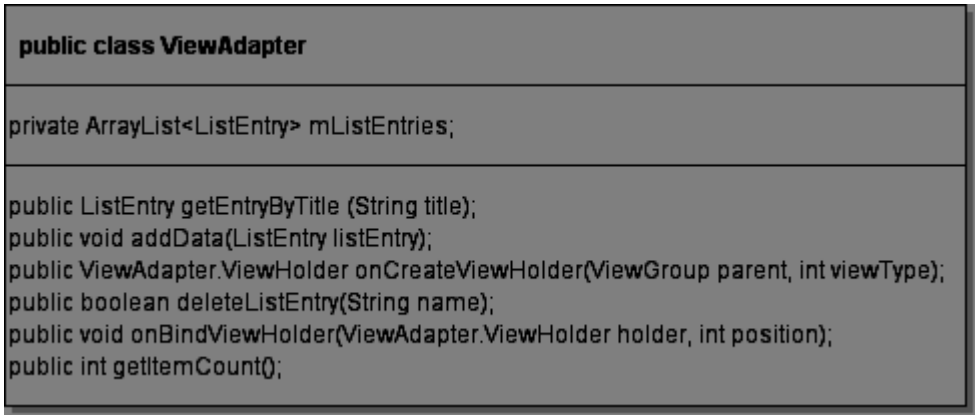

Abbildung 5 - "View Adapter" UML Diagramm

2. Kurzbeschreibung

Die Klasse „**ViewAdapter**“ erbt von der Klasse „RecyclerView.Adapter<ViewAdapter.ViewHolder>“. Diese Klasse kümmert sich um das Befüllen des Layouts „list_entry.xml“ mit den Daten (Bitmap, Titel und Datum). Die Klasse wird für jeden Datensatz neu aufgerufen und ausgeführt.

3. Methoden

Die Methode „**onCreateViewHolder**(ViewGroup, int)“ erzeugt einen ViewHolder und gibt diesen wieder bei Aufrufen der Methode zurück. Ein ViewHolder beschreibt eine View und wie die Daten innerhalb der RecyclerView angeordnet werden müssen.

Die eigentliche Inhalte innerhalb einer View werden in ihre Position gesetzt in der Methode „**onBindViweHolder**(ViewHolder, int). Diese Methode bekommt den ViewHolder mitgegeben und eine Position. Es wird ein „ListEntry“ erstellt und diesem nach und nach die einzelnen Daten also das Bild, der Titel und das Datum hinzugefügt.

Die Methode „**getItemCount**()“ zählt wie viele Einträge die Liste insgesamt hat und gibt diese Anzahl wieder zurück.

7. Klasse – „ViewHolder“

1. “ViewHolder” – UML Diagramm

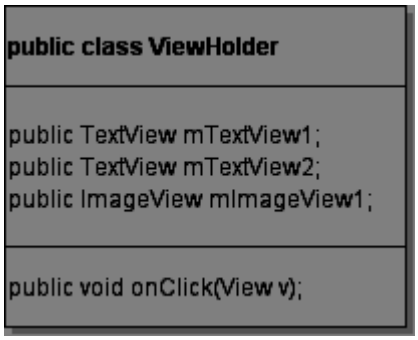

Abbildung 6 - "ViewHolder" UML Diagramm

2. Beschreibung

Die Klasse „**ViewHolder**“ stellt die „Image-View „für das Bitmap und die beiden „Text-Views“ für das Datum und den Titel des jeweiligen Bitmaps für die Klasse „ViewAdapter“ bereit. Dafür werden die „Image-Views“ und die beiden Text-Views als public Variablen instanziiert.

```
public TextView mTextView1;
public TextView mTextView2;
public ImageView mImageView1;
```

```
mTextView1 = (TextView) itemView.findViewById(R.id.textView2);
mTextView2 = (TextView) itemView.findViewById(R.id.textView3);
mImageView1 = (ImageView) itemView.findViewById(R.id.imageView3);
```

Die drei „Views“ werden in eine „Superview“ „ItemView“ zusammengefasst. Des Weiteren wird die „DetailView“ beim Anklicken der „ItemView“ aufgerufen und ausgeführt. Für dieses zu öffnende „DetailView“ wird eine neue „Activity“ gestartet.

```
v.getContext().startActivity(detail_view);
```

8. Klasse – „Settings“

1. „Settings“ – UML Diagramm

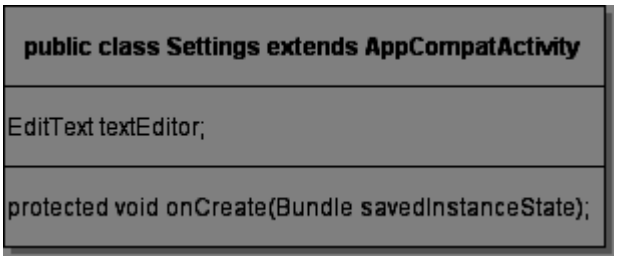

Abbildung 7 - "Settings" UML Diagramm

2. Kurzbeschreibung

Die Klasse „**Settings**“ erbt von „AppCompatActivity“ und überschreibt deren Methode „onCreate“. In dieser Klasse wird der „Settings-Screen“ beschrieben, auf dem der „default filename“ editiert werden kann. Als Attribut enthält sie einen „EditText“ der als Wer „default_Filename“ von MainMenu zugewiesen bekommt.

3. Methoden

Die Methode „**onCreate**“ dient zum Initialisieren der Oberfläche. Zuerst wird die Oberfläche über das XML „R.layout.activity_settings“ gesetzt. Als nächstes wird die „toolbar“ über die Methode „findViewById“ mit dem Parameterwert „R.id.toolbar“ geholt und gesetzt mit „setSupportActionBar“. Das Attribut „textEditor“ wird ebenfalls von „findViewById“ befüllt und wird mit einem „Listener“ versehen. Dieser „OnEditorActionListener“ besitzt die Funktion „onEditorAction“ welche das Verhalten definiert. Falls als „ID“ „EditorInfo.IME_ACTION_DONE“ übergeben wurde, so wird nun der neue „default_Filename“ gesetzt. Dazu werden die „SharedPreferences“ über den „preferences_file_key“ aus dem „MainMenu“ geholt und ein Editor dazu erzeugt. Dieser setzt mit der Funktion „putString“ den neuen Wert. Anschließend werden die „SharedPreferences“ mit „editor.apply()“ aktualisiert. Schließlich wird noch eine Meldung mit „Toast.makeText“ ausgegeben, bei der der neue Name angezeigt wird. Letztlich wird zurückgegeben, ob eine Änderung durchgeführt wurde.

9. Abbildungsverzeichnis