HANSER

Leseprobe

zu

„Apps für Android entwickeln“

von Jan Tittel und Jochen Baumann

ISBN (Buch): 978-3-446-43191-1
ISBN (E-Book): 978-3-446-43315-1

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-43191-1
sowie im Buchhandel

9 Audiodaten aufnehmen, abspielen und die App mit Gesten steuern

In Kapitel 8 haben wir die Beispiel-App um die Möglichkeit erweitert, Bild-Notizen zu erstellen und haben dabei folgende Themen behandelt:

- unterschiedliche Gesten erkennen und verarbeiten
- eine eigene View-Klasse erstellen
- auf dem Bildschirm zeichnen
- Fotos aus der Galerie auswählen und importieren
- Image-Dateien umwandeln und verändern

Die Beispieldateien zu diesem Kapitel finden Sie unter *www.downloads.hanser.de* im Unterordner *scytenotes 0.6*.

In diesem Kapitel werden wir den dritten Typ von Notizen umsetzen. Es handelt sich dabei um Audio-Notizen, also um gesprochenen Text, der aufgenommen und wiedergeben werden kann. Hierbei werden folgende Themen der Entwicklung für Android berücksichtigt:

- Erkennen und Verarbeiten von simplen Gesten
- Audio-Daten aufnehmen und wiedergeben
- Oberflächenelemente aus einem anderen Thread aktualisieren

9.1 Touch Events auswerten mit GestureDetector

Das Layout der *NoteVoiceActivity* werden wir nicht im Einzelnen erläutern, da dies sehr einfach aufgebaut ist. Jedoch wollen wir Ihnen den Aufbau und die Funktionsweise erläutern, um im weiteren Verlauf dieses Abschnitts die Verwendung der Klasse *GestureDetector* zu verstehen.

Wie in Bild 9.1 zu sehen, verfügt die Oberfläche nur über einen Button in der unteren *ActionBar* zum Wechseln von Aufnahme zu Wiedergabe und umgekehrt. Die Steuerung der

Aufnahmen oder Wiedergaben erfolgt durch Touch-Ereignisse auf der Oberfläche, die zu den Gesten zählen. Wir wollen Ihnen damit einen Einstieg in die Möglichkeiten der Steuerung durch Gesten von Android geben. Neben dem Ihnen schon bekannten Interface *OnTouchListener* wird zusätzlich die Klasse *GestureDetector* verwendet, um einfache Gesten, wie Einfach-Tap oder Doppel-Tap, zu erkennen. Die Auswertung der Touch-Ereignisse erfolgt nur im Weiß angezeigten Bereich der Oberfläche. Dazu wurde im Layout ein Kind-Layout des Typs Relative Layout eingefügt und im Code referenziert. Damit ist es weiterhin möglich, das Klick-Ereignis des Buttons in der unteren ActionBar auszuwerten.

Die Activity verfügt über ein gemeinsames Layout für die Wiedergabe und Aufnahme. Der Wechsel von Wiedergabe zu Aufnahme wird durch den Image-Button gekennzeichnet. Im Aufnahmemodus erhält dieser Image-Button als Hintergrundbild das Record-Symbol, im Wiedergabemodus das Symbol für Play. Weiterhin unterscheidet sich die Darstellung der *ProgressBar*: Im Aufnahmemodus läuft der Balken von links nach rechts, und die Anzeige der Zeit in der TextView wird hochgezählt. Im Wiedergabemodus läuft der Anzeigebalken von rechts nach links, und die Spieldauer wird heruntergezählt. Um diesen Effekt zu erreichen, werden die Metadaten der Audio-Datei ausgelesen und die Zeit heruntergezählt.

Die zu erkennenden Gesten sollen nicht auf dem gesamten Layout ausgewertet werden, sondern nur in einem ausgewählten Bereich der Oberfläche. Bei einem einfachen Touch auf die Oberflächen soll die Aufnahme bzw. das Abspielen der Audio-Datei gestartet bzw. gestoppt werden. Erfolgt ein Double Touch, soll in den Modus Aufnahme/Abspielen gewechselt werden.

Um die Touch-Ereignisse auf der Oberfläche auswerten zu können, werden Methoden benötigt, die diese auswerten. Die Klasse *GestureDetector* stellt in Verbindung mit der Komfortklasse *SimpleOnGestureListener* genau diese Funktionen zur Verfügung.

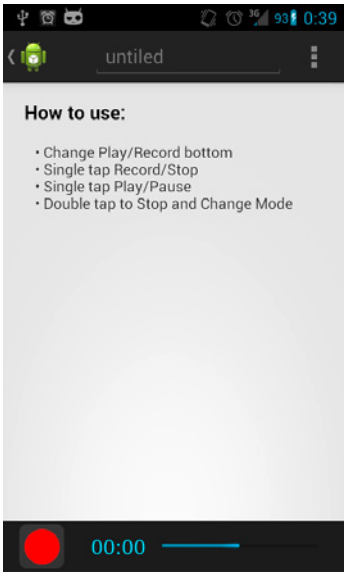

Bild 9.1
Ansicht NoteVoiceActivity

Um die Klasse *GestureDetector* verwenden zu können, muss die Activity-Klasse das Interface *OnTouchListener* implementieren und eine View das Interface binden. Weiterhin muss die Klasse *GestureDetector* initialisiert werden und erwartet als zweiten Parameter einen bereitgestellten Listener, wie in Listing 9.1 dargestellt. Wir verwenden hier die Komfortklasse *SimpleOnGestureListener*, die das Interface *GestureDetector.OnDoubleTapListener* und *GestureDetector.OnGestureListener* schon implementiert.

Listing 9.1 Auszug aus der onCreate()-Methode NoteVoiceActivity

```
protected void onCreate(Bundle savedInstanceState) {

   RelativeLayout touchLayout = (RelativeLayout)
                                findViewById(R.id.note_voice_touch);
   touchLayout.setOnTouchListener(this);
   mGestureDetector = new GestureDetector(this,
                                          simpleOnGestureListener);
```

Bei einem Touch-Ereignis der Oberfläche wird die Callback-Methode *onTouch()* vom Interface *OnTouchListener* aufgerufen und erhält die View und auch das Ereignis als Parameter, die Methode ruft den *GestureDetector* auf und leitet das Ereignis an diesen weiter (siehe Listing 9.2).

Listing 9.2 Überschriebene Methode onTouch()

```
@Override
public boolean onTouch(View v, MotionEvent event) {
   this.mGestureDetector.onTouchEvent(event);
   return true;
}
```

Der Listener des *GestureDetectors* nimmt das Ereignis entgegen und ruft die entsprechende Methode des erkannten Ereignisses auf. Der in Listing 9.3 deklarierte *SimpleOnGestureListener* enthält nur zwei überschriebene Methoden. Dies sind jedoch nicht alle Methoden, die die Klasse *SimpleOnGestureListener* zur Verfügung stellt. In der App werden allerdings nur der einfache Tap und der Doppel-Tap benötigt. Weitere Informationen zur Klasse finden Sie unter *http://developer.android.com/reference/android/view/GestureDetector.SimpleOnGestureListener.html.*

Listing 9.3 Listener zum Erkennen einfacher Gesten

```
SimpleOnGestureListener simpleOnGestureListener =
                                     new SimpleOnGestureListener(){

   @Override
   public boolean onDoubleTap(MotionEvent e) {
      switchMediaMode();
      return true;
   }

   @Override
   public boolean onSingleTapConfirmed(MotionEvent e) {
       onStartStop();
       return true;
    }
};
```

9.2 Audios aufnehmen und abspielen

Mit den Klassen *MediaRecorder()* und *MediaPlayer()* werden die Audiodaten aufgezeichnet bzw. wiedergegeben. Die Activity verfügt über zwei Modi: den Aufnahme- und Wiedergabemodus. Dies wird durch das *enum* `MEDIA_MODE` repräsentiert. Um zu prüfen, ob eine Aufnahme oder Wiedergabe aktuell ausgeführt wird, werden die Statusvariablen *onRecording* und *onPlaying* des Typs *boolean* abgefragt, die den Wert *false* oder *true* zurückliefern.

Der Wechsel von Aufnahme- zu Wiedergabemodus und umgekehrt erfolgt durch einen Doppel-Tap auf die Oberfläche oder durch Betätigen des Image-Buttons in der unteren Bedienleiste. Durch einen Einfach-Tap auf die Oberfläche wird je nach Modus die Aufnahme oder Wiedergabe gestartet. Bei der Wiedergabe kann zusätzlich durch einen Einfach-Tap diese unterbrochen bzw. fortgesetzt werden.

Im folgenden Abschnitt befassen wir uns zuerst mit dem Aufzeichnen von Audio-Daten, um diese später wiedergeben zu können.

Die Klasse *MediaRecorder()* unterstützt leider nicht das Pausieren von Aufnahmen, sodass beim Stoppen der Aufnahme und erneutem Starten eine neue Datei erstellt wird.

9.2.1 Audio-Notizen erstellen

Die Aufnahme der Audio-Dateien erfolgt durch die Methode *recordTempFile()* aus Listing 9.4. Die Aufnahme der Audio-Daten erfolgt zunächst in einer temporären Datei im Cache Verzeichnis der App. Erst wenn der Benutzer die Notiz speichert, wird diese in das Verzeichnis *ScyteNotes/voice* kopiert. Jede App kann ein eigenes Cache-Verzeichnis nutzen, das Verzeichnis befindet sich unter *\storage\sdcardX\Android\eu.scyte.notes\cache*.

Die Methode *recordTempFile()* wird aufgerufen, wenn das *enum* `MEDIA_MODE` den Wert RECORDING hat.

Zunächst erfolgt die Überprüfung, ob das File-Objekt *tempFile* schon existiert. Ist diese Bedingung in der *if*-Anweisung erfüllt, wird die bestehende temporäre Datei gelöscht. Beachten Sie, dass jede neu angelegte, temporäre Datei eine andere Id hat. Die Erstellung von temporären Dateien muss in einem *try/catch*-Block gekapselt werden.

Mit der Methode *.createTempFile* der Klasse *File* wird zunächst ein neues *File*-Objekt erstellt, dessen erster Parameter ein Präfix des Dateinamens ist, dem weitere Zeichen angehängt werden. Der zweite Parameter ist die Dateiendung, und der letzte Parameter legt das Cache-Verzeichnis fest.

Im nächsten Schritt wird ein *MediaRecorder*-Objekt erzeugt und initialisiert. Um den Media Recorder verwenden zu können, müssen die Source, hier das Mikrofone, das Output-Format, der Encoder und die Ausgabedatei definiert sein. Weiterhin haben wir die maximale Aufnahmezeit auf 59 Minuten und 58 Sekunden begrenzt, da die TextView der Zeitanzeige nur maximal 59 Minuten und 59 Sekunden darstellen kann.

Nachdem nun der MediaRecorder konfiguriert ist, wird im nächsten Schritt mit *.prepare()* der MediaRecorder für die Verwendung vorbereitet. Wäre die Konfiguration fehlerhaft, würde eine *IllegalStateException* ausgelöst. Nach erfolgreicher *.prepare()*-Methode startet

jetzt der MediaRecorder. Im nächsten Schritt wird der *mRecorderThread* initialisiert und mit *.start()* gestartet. Der *mRecorderThread* führt Methoden aus, um den Fortschrittsbalken der *ProgressBar* sowie die Zeitanzeige jede Sekunde zu aktualisieren. Zum Abschluss wird der Statusvariablen *onRecording* noch der Wert *true* zugewiesen.

Die Verwendung des MediaRecorders kann zwei verschiedene Ausnahmen auslösen. Zum einen kann die Konfiguration fehlerhaft sein und einen fehlerhaften Status wie den *catch*-Block mit der Ausnahme *IllegalStateException* verursachen, oder der Zugriff auf das Datei-Objekt *tempFile* schlägt fehl und löst eine Ausnahme aus.

Listing 9.4 Methode zum Aufnehmen temporärer Audio-Dateien

```
private void recordTempFile() {
   try {
        if(tempFile != null && tempFile.canRead()){
        tempFile.delete();
        }
        tempFile = File.createTempFile("record", ".3gp",
        getExternalCacheDir());
   } catch (IOException e) {
        e.printStackTrace();
   }

   mRecorder = new MediaRecorder();
   mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
   mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
   mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
   mRecorder.setMaxDuration(3598000);
   mRecorder.setOutputFile(tempFile.getAbsolutePath());
   try {
        mRecorder.prepare();
        mRecorder.start();
        mRecorderThread = new Thread(mRecorderRunnable);
        mRecorderThread.start();
        onRecording = true;
   } catch (IllegalStateException e) {
        onRecording = false;
        e.printStackTrace();
   } catch (IOException e) {
        onRecording = false;
        e.printStackTrace();
   }
}
```

9.2.2 Audio-Notiz abspielen

Das Abspielen von Audio-Dateien erfolgt durch die Methode *playFile()*. Diese Methode erhält als Parameter die Audio-Datei, die abgespielt werden soll. Die eigentliche Wiedergabe der Datei erfolgt durch das Objekt *mPlayer* der Klasse *android.media.MediaPlayer*. Mit dem Objekt *retriever* des Typs *android.Media.MediaMetadataRetriever* wird die Abspielzeit der Audio-Datei ermittelt. Die Meta-Daten der Audio-Datei können neben der Abspielzeit in Millisekunden, den Künstler, das Album, die Bit-Rate etc. enthalten. Der Zugriff auf die entsprechenden Informationen erfolgt per Integer Key. Jeder verfügbare Key ist als Konstante im System gespeichert. Die einzelnen Werte der Meta-Daten sind vom Typ String

sowie auch die Abspieldauer, sodass eine Konvertierung in den Typ Integer notwendig ist und auf ganze Sekunden aufgerundet wird, da die TextView für die Anzeige der Zeit nur Minuten und Sekunden darstellt.

Nachdem nun die Abspielzeit ermittelt und der Fortschrittsbalken der *ProgressBar* auf 100 Prozent gestellt wurde, erfolgt die Initialisierung des MediaPlayer-Objekts *mPlayer*.

Als Erstes erfolgt die Konfiguration des Audio Stream-Typs durch die Klasse *AudioManager*. Mit der nächsten Anweisung erfolgt die Erstellung des *setOnCompletionListener*. Dieser überwacht, ob der MediaPlayer die Wiedergabe abgeschlossen hat und ruft am Ende der Wiedergabe die Methode *onCompletion()* auf. In der Methode erhält die Statusvariable *onPlaying* des Players den Wert *false*. Es wird eine Meldung angezeigt, dass die Wiedergabe beendet ist, und die MediaPlayer-Instanz wird gelöst.

Im nächsten Schritt erfolgt eine Sicherheitsprüfung mit der *if*-Anweisung dahingehend, ob die als Parameter erhaltene Referenz auf die Audio-Datei gelesen werden kann. Bei Erfolg wird zunächst der Player durch *.reset()* einmal zurückgesetzt. Dann erfolgt die Zuweisung der Audio Source und weiter die Vorbereitung des Players durch *.prepare()*. Im nächsten Schritt wird die Wiedergabe *mPlayer.start()* gestartet, und die Statusvariable *onPlaying* erhält den Wert *true*.

Wie auch beim MediaRecorder können hier die Ausnahmen *IllegalStateException* und *IOException* auftreten.

Listing 9.5 Methode zum Abspielen von Audio-Dateien

```
public void playFile(File file) {

   MediaMetadataRetriever retriever = new MediaMetadataRetriever();
   retriever.setDataSource(file.getAbsolutePath());
   String meta = retriever
     .extractMetadata(MediaMetadataRetriever.METADATA_KEY_DURATION);
   currentPlaySecond = playTime = (int)
                       Math.round(Double.parseDouble(meta) / 1000);
   progressBar.setProgress(100);
   mPlayer = new MediaPlayer();
   mPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
   mPlayer.setOnCompletionListener(new OnCompletionListener() {
      public void onCompletion(MediaPlayer arg0) {
         onPlaying = false;
         Toast.makeText(getApplicationContext(), "Play End",
                        Toast.LENGTH_LONG).show();
         if (mPlayer != null) {
            mPlayer.release();
            mPlayer = null;
         }
       }
     });
   try {
      if (file.canRead()) {
         mPlayer.reset();
         mPlayer.setDataSource(file.getAbsolutePath());
         mPlayer.prepare();
         mPlayer.start();
         onPlaying = true;
   } catch (IllegalStateException e) {
         onPlaying = false;
```

```
            e.printStackTrace();
        } catch (IOException e) {
            onPlaying = false;
            e.printStackTrace();
        }
    }
```

9.3 Threading

Bisher ist die Anwendung so erstellt, dass alle Methoden und Aufrufe im UI-Thread der App ausgeführt werden. Diese Vorgehensweise hat den Nachteil, dass sehr schnell ein so genannter ANR *(application not responsible)* auftreten kann. Die Android-Plattform erwartet, dass dem Anwender die Oberfläche einer Activity innerhalb von 5 bis 10 Sekunden angezeigt wird. Dauert die Erstellung der Activity länger, so reagiert das System mit einer Meldung: „Um so genannte ANRs zu vermeiden, können Sie Threads verwenden“.

Die Verwendung eines eigenen Threads in dieser Activity hat jedoch einen anderen Grund. Die *ProgressBar* und die Zeitanzeige sollen im Sekundentakt aktualisiert werden. Diese Aktualisierungen erfolgen in einer *while*-Schleife. Der ausführende Thread wird durch *Thread.sleep(1000)* bei jedem Durchlauf der Schleife für eine Sekunde angehalten. Dies hätte zur Folge, dass die Activity in dieser Sekunde auf keine Klick- oder Touch-Ereignisse reagiert.

Die Verwendung von Threads in Android entspricht vollständig der Threads in Java. Um Threads verwenden zu können, muss das Interface *Runnable* implementiert sein. Alle Methoden, die innerhalb des Threads ausgeführt werden sollen, befinden sich in der *void run()*-Methode. Die *while*-Schleife innerhalb der *run*-Methode sorgt dafür, dass der Thread so lange ausgeführt wird, bis die Bedingung in der *while*-Schleife nicht mehr erfüllt wird. In unserem Fall, wenn *onRecording* den Wert *false* annimmt. Jeder Thread kann nur einmal gestartet und verwendet werden. Wurde ein Thread einmal gestartet, muss eine neue Instanz erzeugt werden. Der Thread wird durch `Thread t = new Thread(Runnable)` erzeugt, mit `t.start()` wird er gestartet. Die Methode *Thread.sleep(...)* unterbricht ihn für die angegebene Zeit in Millisekunden.

Es ist nicht möglich, von einem separaten Thread auf den UI-Thread der Oberfläche zuzugreifen, jedoch stellt Java mit der *.post()-Methode* von UI-Elementen eine Möglichkeit zur Verfügung, trotzdem Nachrichten an ein UI-Element des Main- bzw. UI-Threads zu versenden.

Der Thread für die Aktualisierung der Zeitanzeige und ProgressBar wird mit der Aufnahme gestartet. Zunächst werden zwei einfache Zählvariablen mit ihren Startwerten initialisiert. Die *while*-Schleife wird so lange ausgeführt, bis *onRecording* den Wert *false* annimmt oder die Recorder-Instanz *null* ist. Innerhalb der *while* Schleife werden die Zählvariablen immer um eins erhöht. Hat der Zähler der *ProgressBar* den Wert 100 erreicht, fängt er wieder bei 0 an zu zählen. Der Thread wird bei jedem Durchlauf der Schleife für 1 Sekunde angehalten. Bei jedem Durchlauf erhalten die innerhalb der Schleife erstellten finalen Value-Variablen den Wert der Zählervariablen zugewiesen.

Mit dem Aufruf der *.post()*-Methode der UI-Elemente *TextView* und *ProgressBar* erfolgt die Aktualisierung der Anzeige.

Durch einen Tap auf die Oberfläche wird der MediaRecorder beendet, und *onRecording* erhält den Wert *false*. Beim nächsten Eintreten in den Kopf der *while*-Schleife beendet sich der Thread, da die Bedingung der *while*-Schleife nicht mehr erfüllt ist.

Listing 9.6 RecorderRunnable zur Aktualisierung der ProgressBar und Zeitanzeige

```
mRecorderRunnable = new Runnable() {
   @Override
   public void run() {
      int countProgress = 0;
      int countSeconds = 1;
      while (mRecorder != null && onRecording) {
         try {
             if(countProgress == 100){
                countProgress = 0;
             }
            Thread.sleep(1000);
            countProgress++;
            countSeconds++;
         } catch (InterruptedException e1) {
           e1.printStackTrace();
         }
         final int progressValue = countProgress;
         final int secondsValue = countSeconds;
         timeTextView.post(new Runnable() {
            @Override
            public void run() {
            setDisplayTime(secondsValue);
            }
         });
         progressBar.post(new Runnable() {
            @Override
            public void run() {
              setProgressBarValue(progressValue);
            }
          });
       }
     }
   };
```

9.4 Zwischenstand der App (Version 0.6)

In diesem Kapitel haben wir die Verwendung der Klassen *MediaRecorder* und *MediaPlayer* erläutert, ebenso erhielten Sie einen Einstieg in die Verwendung von Gesten zur Steuerung von Android. Zum Abschluss zeigten wir, wie Sie Threads benutzen können, um zu verhindern, dass die Oberfläche der App nicht mehr reagiert.

In Kapitel 10 werden Sie lernen, wie Datenbanken in Android verwendet werden.