

jan TITTEL
jochen BAUMANN



APPS FÜR ANDROID ENTWICKELN

AM BEISPIEL EINER REALEN APP

HANSER

Tittel/Baumann

Apps für Android entwickeln



Bleiben Sie auf dem Laufenden!

Der Hanser Computerbuch-Newsletter informiert Sie regelmäßig über neue Bücher und Termine aus den verschiedenen Bereichen der IT. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter www.hanser-fachbuch.de/newsletter

Jan Tittel
Jochen Baumann

Apps für Android entwickeln

Am Beispiel einer realen App

HANSER

Die Autoren:

Jan Tittel, Essen

Jochen Baumann, Oberhausen

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso übernehmen Autor und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürfen.



Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2014 Carl Hanser Verlag München, www.hanser-fachbuch.de

Lektorat: Sieglinde Schärl

Copy editing: Kathrin Powik, Lassan

Herstellung: Irene Weilhart

Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München

Umschlagrealisation: Stephan Rönigk

Gesamtherstellung: Kösel, Krugzell

Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702

Printed in Germany

print-ISBN: 978-3-446-43191-1

e-book-ISBN: 978-3-446-43315-1

Inhalt

1 Einführung	1
1.1 Die Android-Plattform	2
1.2 An wen richtet sich dieses Buch?	3
1.3 Buchaufbau und verwendete Technologien	3
1.4 Vorstellung des Beispielprojekts	4
1.5 Danksagung	9
2 Einrichten der Arbeitsumgebung	11
2.1 Installation von Java	11
2.2 Einrichtung und Konfiguration von Eclipse und ADT	14
2.3 Geräte für die Entwicklung einrichten	23
3 Schnelleinstieg in Eclipse und Java	27
3.1 Die erste App mit Eclipse und dem ADT erstellen	27
3.1.1 Ein neues Projekt in Eclipse anlegen	28
3.1.2 Erster Einstieg in Eclipse mit ADT	32
3.1.3 Die Oberfläche der App anpassen (XML-Layout)	35
3.1.4 Der App Funktionalität geben	41
3.1.5 Die Struktur von Android-Projekten	43
3.2 Die App im Emulator und auf einem Gerät testen	46
3.2.1 Emulator definieren	47
3.2.2 Tasturbefehle für den Android-Emulator	49
3.2.3 Die App auf einem Emulator oder Gerät starten	49
3.3 Crash-Kurs in Java	52
3.3.1 Werte einer Variablen zuordnen	54
3.3.2 Bezeichner und Schlüsselwörter in Java	54
3.3.3 Klassen und Objekte in Java	55
3.3.4 Objekte erzeugen und initialisieren	57
3.3.5 Methoden von Objekten aufrufen	57
3.3.6 Auf Eigenschaften von Objekten zugreifen	58
3.3.7 Abstrakte Klassen und Methoden	59

3.3.8 Interfaces (Schnittstellen)	61
3.3.9 Java Packages (Pakete)	62
3.3.10 Logging	62
3.3.11 Kommentare	62
3.3.12 Ablaufsteuerung mit einfacher Verzweigung	63
3.3.13 Ablaufsteuerung mit Mehrfachverzweigungen	64
3.3.14 Wiederholungen mit Schleifen	64
3.3.15 Fehlerbehandlung	65
4 Grundlagen von Layouts, Views, Komponenten und Intents:	
Erste Oberflächen erstellen	67
4.1 Zielsetzung	67
4.2 Layouts, Views und Komponenten	67
4.2.1 Layouts	68
4.2.2 Views und Widgets	69
4.2.3 Bausteine einer Android-Anwendung	69
4.3 Layouts und Activities	75
4.3.1 Layouts erstellen	75
4.3.2 Activities	80
4.4 Ereignisse und Intents	81
4.4.1 Auf Klick-Ereignisse der Oberfläche reagieren	82
4.4.2 Mit Intents eine andere Activity aus der aktuellen Activity aufrufen	82
4.5 Activities im Manifest registrieren	84
5 Menüs und Ressourcen	85
5.1 Menüs	85
5.1.1 Optionsmenü erstellen	85
5.2 String-Ressourcen	88
5.2.1 Eine String-Ressource anlegen und referenzieren	88
5.3 Drawable-Ressourcen	89
5.3.1 Drawable-Ressourcen verwenden	89
5.3.2 ActionBar-Icons erstellen und verwenden	90
5.3.3 xEffekte durch Drawables	91
6 Eigene Klassen, Listen und Adapter	95
6.1 Eigene Klassen erstellen	95
6.2 Enumerationen	97
6.3 Arrays, Listen und Adapter	98
6.3.1 Einfache Strings in Spinner und Liste anzeigen	99
6.3.2 Eigene Objekte in einer ListView anzeigen	101
6.3.3 ListItem undListAdapter für eigene Objekte	103
6.3.4 ListItem auswählen	107
6.3.5 LogCat verwenden	108

6.4	Mit Dateien arbeiten	109
6.4.1	Datei im Dateisystem speichern	110
6.4.2	Inhalte von Verzeichnissen auflisten und in ListView anzeigen	117
6.4.3	Dateien löschen	122
6.4.4	Dateien lesen	124
6.5	Zwischenstand der App (Version 0.3)	128
7	ActionBar, WebView und E-Mail	129
7.1	ActionBar erweitern und Funktionen nutzen	129
7.1.1	Eigenes Layout-Element in der ActionBar verwenden und App-Icon für die Navigation nutzen	129
7.2	E-Mail mit Anhang versenden	133
7.3	Lokale und externe Webseiten anzeigen	136
7.3.1	Erstellen und Anpassen der WebView Activity	136
7.4	Zwischenstand der App (Version 0.4)	140
8	Fragments, Touch Events und Canvas	141
8.1	Fragments	141
8.1.1	Fragments, Drawable-Ressourcen	142
8.2	Image View erweitern, Canvas und Touch Events	150
8.3	Activity for Result und Grafikbearbeitung	156
8.3.1	Mit Activity for Result den Pfad zum Bild ermitteln und sich das Ergebnis in einem Toast anzeigen lassen	156
8.3.2	Ein Bild in ein Bitmap umwandeln	157
8.4	Zwischenstand der App (Version 0.5)	161
9	Audiodaten aufnehmen, abspielen und die App mit Gesten steuern	163
9.1	Touch Events auswerten mit GestureDetector	163
9.2	Audios aufnehmen und abspielen	166
9.2.1	Audio-Notizen erstellen	166
9.2.2	Audio-Notiz abspielen	167
9.3	Threading	169
9.4	Zwischenstand der App (Version 0.6)	170
10	Dialog-Fragments und Datenbank	171
10.1	Dialog-Fragments	171
10.1.1	Klasse für Erinnerungen, Reminder	177
10.2	Datenbank in Android verwenden	179
10.2.1	Datenbank erstellen	180
10.2.2	Datensätze hinzufügen, ändern und löschen	183
10.2.3	Alle Datensätze einer Tabelle auslesen	186
10.2.4	Datenbankzugriff per Kommandozeile oder Eclipse Plug-in	187

10.3 Daten aus der Datenbank anzeigen	189
10.4 Zwischenstand der App (Version 0.7)	190
11 Google Maps Api V2 und LocationService	191
11.1 Vorbereitungen zur Verwendung von Google Maps Api V2	191
11.1.1 API-Key und Rechte im Manifest anpassen	195
11.2 Ortsbestimmung mit dem LocationService	198
11.3 Eigene Activity for Result für die Map	201
11.4 Zwischenstand der App (Version 0.8)	208
12 Zeit- und ortsbasierte Erinnerungen, lokale Notifications	209
12.1 Zeitbasierte Erinnerungen mit dem AlarmManager	209
12.2 BroadcastReceiver und NotificationManager	211
12.3 Ortsbasierte Erinnerungen mit ProximityAlerts	212
12.4 Zwischenstand der App (Version 0.9)	214
13 Lokalisierung, Icons und Startbilder – Vorbereitung für die Veröffentlichung im Play Store	215
13.1 Die App lokalisieren	215
13.1.1 String-Ressourcen lokalisieren	216
13.2 Icons und Bilder	216
13.3 Der letzte Feinschliff	217
13.3.1 Mit Android Lint den Code verbessern	220
14 Veröffentlichung einer App im Play Store	223
14.1 App mit eigener Signatur exportieren	223
14.2 App in den Play Store hochladen	226
14.3 Store-Eintrag erstellen und Icons und Screenshots verwalten	228
14.4 Preisgestaltung und Vertrieb	228
Index	231

1

Einführung

Der Buchmarkt hält bereits einiges an Literatur zur App-Entwicklung für Smartphones bereit. Angefangen vom einfachen, dünnen Buch für Einsteiger bis hin zum dicken Wälzer und Nachschlagewerk für Profis. Warum also ein weiteres Buch?

Wir sind selbst seit Jahren als Entwickler tätig und haben mit dem Aufkommen des Booms selbstverständlich auch mit der App-Entwicklung für Smartphones begonnen. Dabei haben wir insbesondere zwei Schwachstellen ausgemacht.

Zum einen holen die meisten Bücher oder Tutorien sehr weit aus und beginnen mit den einfachsten Dingen. Ein Großteil der Entwickler startet mit einer App allerdings nicht sein erstes Projekt, sondern verfügt bereits über umfangreiche Kenntnisse in anderen Programmiersprachen. Für viele Entwickler ist daher ein Schnelleinstieg vollkommen ausreichend.

Zum anderen ist es für eine erfolgreiche App unerlässlich, diese für mehrere Plattformen, zumindest für die beiden derzeit erfolgreichsten (iOS und Android), zu entwickeln. Zusätzlich vermissen wir oftmals die Umsetzung von realen Beispielen, da es dabei dann doch einige Aspekte zu berücksichtigen gilt, die in einer rein technisch funktionalen Beschreibung der App-Entwicklung untergehen.

So wurde die Idee geboren, eine tatsächlich existierende Beispiel-App für mehrere Plattformen zu entwickeln und daraus gleichzeitig ein Buch – einen Schnelleinstieg für IT-Profis – zu machen. Durch dieses Konzept ist es für Sie als Leser möglich, sich an ein und demselben Beispiel einen Überblick über die verschiedenen Plattformen zu verschaffen und gleichzeitig mit möglichst wenig Aufwand die Unterschiede bei der Entwicklung herauszuarbeiten. Das vorliegende Buch konzentriert sich daher neben den notwendigen Grundlagen auf die Umsetzung eines umfangreicheren Beispiels.

Parallel erscheint dieses Buch auch für iOS (*Apps für iOS entwickeln*, Carl Hanser Verlag 2013, ISBN 978-3-446-43192-8). Weitere Plattformen sind bereits in Planung, deren konkrete Umsetzung hängt aber auch von der Entwicklung des Marktes ab, sodass wir an dieser Stelle keine falschen Versprechungen machen möchten.

■ 1.1 Die Android-Plattform

Android ist noch eine verhältnismäßig junge Plattform für Smartphones und andere mobile Geräte, wenn man bedenkt, wie lange mobile Plattformen schon eingesetzt werden. Denken Sie beispielsweise an Symbian, Java embedded, Windows CE/Windows Mobile oder BlackBerry OS.

Der Ursprung von Android liegt in dem von Andy Rubin Ende 2003 in Palo Alto gegründeten Unternehmen Android Inc., das Google im Jahr 2005 vollständig übernahm. Android Inc. entwickelte Software für Mobiltelefone und favorisierte standortbezogene Dienste.

Viel mehr war von Android Inc. nicht bekannt und auch Google hüllte sich nach der Übernahme in Schweigen. Die offizielle Lesart war: „Man habe Android Inc. wegen der talentierten Ingenieure und der Technologie übernommen.“ Letztendlich stimmte dies ja mehr oder weniger auch. Bis 2007, genauer gesagt bis zum 5. November dieses Jahres, nahm der normale Nutzer kaum Notiz von diesen Aktivitäten seitens Google. Doch an diesem Tag trat Google an die Presse und gab die Gründung der OHA (Open Handset Alliance) bekannt. Das Konsortium hatte 47 Gründungsmitglieder aus verschiedenen Branchen. Neben bekannten Geräteterstellern und Netzbetreibern wie HTC und T-Mobile gehörten auch Software- und Marketingfirmen sowie Unternehmen der Halbleiterindustrie zu den Gründungsmitgliedern. Eine Woche nach Gründung der OHA erfolgte die Veröffentlichung des ersten offiziellen Android SDK.

Der Launch des ersten Smartphones mit Android wurde allerdings erst für das Jahr 2008 angekündigt. Seit dieser ersten Version wurde Android stetig verbessert. Während anfangs der Fokus noch auf der Funktionalität und weniger auf dem Design lag, so ist die Version 4.0 (*Ice Cream Sandwich*) auch in Sachen Design ein echter Meilenstein. Google reagierte auf die zunehmende Fragmentierung der Plattform, auf die Erfahrungen mit der Version 3.0 (*Honeycomb*), die Tablets vorbehalten war, und erstellte eine verbesserte Version für alle Plattformen (Smartphones, Tablets und TV). TV findet in diesem Buch jedoch keine Beachtung.

Seit *Ice Cream Sandwich* ist es nun viel einfacher, eine App für Smartphones und Tablets zu erstellen, die auf allen Geräten so aussieht und funktioniert, wie man es erwartet, wenn man diese im Standard erstellt hat.

Ein Problem war unter anderem, dass viele Hardware-Hersteller die Android-Oberfläche anpassten und mit eigenen UIs, wie HTC Sense, Samsung TouchWiz, Motorola Motoblur oder Sony, Codename: „Rachel“, auslieferten. Es konnte nun passieren, dass Apps oder Widgets, die auf Standardelementen von Android basieren, auf diesen Geräten anders aussahen oder sogar nicht funktionierten. Dieses Problem besteht seit Version 4.0 nun nicht mehr. Um dieses Problem in den Griff zu bekommen, führte Google das *Holo Theme* und die *Action-Bar* ein, die jetzt auf jedem Gerät verfügbar sind.

Die Version 4.2 (*Jelly Bean*) brachte wieder viele Verbesserungen, und Google legte hier den Fokus der Verbesserungen auf die User Experience und die Stabilität.

Dazu zählen unter anderem folgende Punkte:

- UIs sind noch flexibler zu erstellen und somit ansprechender.
- Verbesserte Browser und App-Widgets
- USB-Audio ist neu hinzugekommen.

- Die DalvikVM wurde stabiler und performanter.
- Ein neuer Bluetooth Stack wurde integriert.
- NFC wurde um Dateiaustausch erweitert.
- Multi-User-Unterstützung

Weitere Informationen zur aktuellen Version erhalten Sie unter <http://www.android.com>.

■ 1.2 An wen richtet sich dieses Buch?

Dieses Buch richtet sich an Entwickler, die bereits grundlegend mit der modernen objekt-orientierten Programmierung vertraut sind. Vorkenntnisse in Java sind nicht notwendig. Hierfür haben wir einen kleinen Crash-Kurs in Java mit in das Buch aufgenommen. Der Schwerpunkt liegt dabei allerdings auf der Syntax und den Grundlagen von Java, die für das weitere Verständnis hilfreich sind. Es wird nicht auf Fragen für Einsteiger eingegangen, worum es sich beispielsweise bei der objektorientierten Programmierung überhaupt handelt. Dieses allgemeine Wissen setzen wir voraus. Außerdem sollten Sie über gute Computerkenntnisse verfügen, da Eclipse und das ADT manchmal Ihre Unterstützung benötigen.

Intention dieses Buches ist es, Ihr vorhandenes allgemeines Programmierwissen auf Android zu übertragen und Ihnen somit einen schnellen und kompakten Einstieg in die App-Entwicklung zu bieten. Die Entwicklung unter Android hat eine geringe Einstiegshürde und sorgt gleich am Anfang für eine steile Lernkurve.



HINWEIS: Für absolute Anfänger und Einsteiger in die Anwendungsentwicklung, ohne Vorkenntnisse in irgendeiner objektorientierten Programmiersprache, ist dieses Buch wenig geeignet, da allgemeine Kenntnisse vorausgesetzt werden.

■ 1.3 Buchaufbau und verwendete Technologien

Nach dieser Einführung wird zunächst die Installation und Konfiguration der Entwicklungsumgebung Eclipse und des ADT-Plug-ins beschrieben. Wir beschränken uns auf die Erläuterung für ein Windows 7-System. Anschließend erhalten Sie einen kurzen Überblick zu Eclipse und den Android Development Tools.

In Kapitel 3 erhalten Sie einen Schnelleinstieg in Java und eine kurze Einführung in Eclipse. Wir haben uns für Eclipse als IDE entschieden, da diese derzeit noch, als favorisiertes Entwicklungssystem von Google angesehen, als Download angeboten wird, und kostenlos sowie für alle Plattformen verfügbar ist. Eine gute Alternative wäre IntelliJ IDEA von JetBrains.

Zum Zeitpunkt der Drucklegung dieses Buches arbeitete Google noch an der neuen Entwicklungsplattform *Android Studio*, die auf IntelliJ IDEA basiert.

Die ersten Kapitel haben wir bewusst mit zahlreichen Screenshots versehen, damit sich auch die Leser unter Ihnen problemlos zurechtfinden, die bislang kaum Erfahrungen mit Eclipse und Java besitzen, und die sich erstmalig mit der Entwicklung unter Eclipse auseinandersetzen.

Die weiteren Kapitel zeigen anhand eines umfangreichen und realen Beispielprojekts die einzelnen Schritte der App-Entwicklung für Android. Das Beispielprojekt umfasst unter anderem folgende Technologien und Features:

- Benutzeroberflächen erstellen
- Mit der ActionBar arbeiten
- Eigene Navigationselemente erstellen
- Eigene Klassen erstellen
- Daten zwischen Activities übergeben
- Texte erstellen und bearbeiten
- Listen darstellen und bearbeiten
- Toasts (Bildschirmnachrichten) einblenden
- Dateien lesen und schreiben sowie mit dem Dateisystem arbeiten
- Datenbanken erstellen und nutzen
- Audio aufnehmen und abspielen
- Standordienste und Karten verwenden, inklusive Reverse Geocoding
- Notifications nutzen
- Lokalisierung der App für mehrere Sprachen
- Fotos aus der Galerie nutzen

Um zu verstehen, was das Ziel unserer Aufgabe ist, wird die App im folgenden Abschnitt kurz vorgestellt, sodass Sie nicht ins kalte Wasser springen müssen und erst am Ende den Sinn und Zweck der beschriebenen Schritte verstehen.

■ 1.4 Vorstellung des Beispielprojekts

Das Beispielprojekt, welches wir im Verlauf dieses Buches erstellen werden, ist eine App für die Erstellung von Notizen in Form von Text, Bild oder Audio.



Den Quellcode der App finden Sie unter www.downloads.hanser.de. Außerdem können Sie sich das fertige Projekt aus dem Google Play Store herunterladen und installieren. Suchen Sie dazu einfach nach *scyte notes*. Weitere Informationen zum Projekt finden Sie auf der dazugehörigen Webseite www.scyte.eu.

Bild 1.1 zeigt die Startseite der App mit der Auflistung aller Notizen.

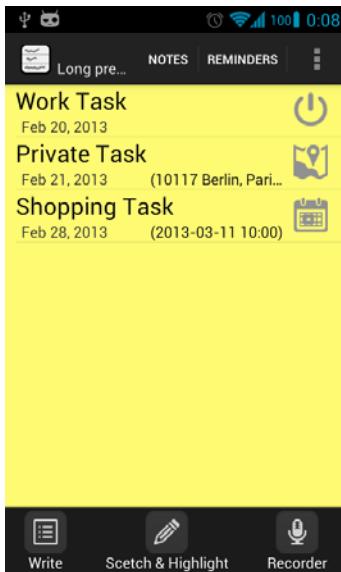


Bild 1.1
Listenansicht aller Notizen

Durch Auswahl des Notiz-Typs und Hinzufügen beziehungsweise Auswählen einer vorhandenen Notiz, gelangt der Benutzer in die spezifische Bearbeitungsansicht der Notiz. Das Beispiel der Bearbeitungsansicht einer Text-Notiz ist in Bild 1.2 zu sehen. Die Bearbeitungsmöglichkeiten sind bewusst einfach gehalten, alle Notizen werden in gängigen Formaten, wie .txt für Text-Notizen, .jpg für Bild-Dateien und .3gp für Audio-Notizen, gespeichert. Somit steht dem Austausch über Plattformgrenzen nichts im Wege, und die Notizen können einfach als E-Mail versendet werden.

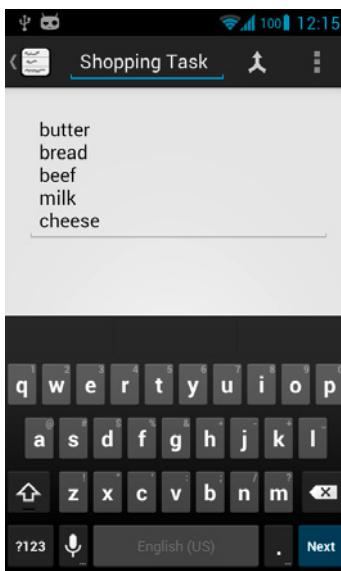
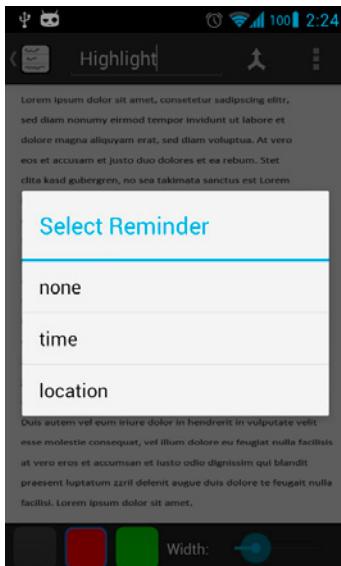


Bild 1.2
Bearbeitungsansicht für Text-Notizen

Nach dem erstmaligen Speichern einer neuen Notiz über das Options-Menü **SAVE**, erscheint ein Auswahldialog, ob eine zeit- oder ortsbasierte Erinnerung oder keine Erinnerung eingerichtet werden soll, wie in Bild 1.3 dargestellt. Außerdem erscheint dieser Auswahldialog, wenn in der Startansicht auf das Icon einer Notiz ohne Erinnerung geklickt wird, wie in Bild 1.4 zu sehen. Pro Notiz kann jeweils nur eine Erinnerung eingerichtet werden. Im Gegensatz zu den Notizen selbst, werden die Erinnerungen nicht in Form von Dateien, sondern in einer Datenbank gespeichert.

**Bild 1.3**

Auswahl dialog für Erinnerungsart beim Speichern

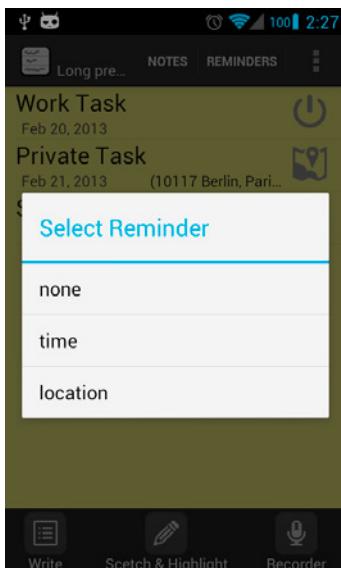


Bild 1.4 Auswahl dialog für Notiz ohne Erinnerung in der Startansicht

Nach der Auswahl der Erinnerungsart öffnet sich der eigentliche Dialog für die Erinnerungen. In Bild 1.5 sehen Sie den Dialog der zeitbasierten und in Bild 1.6 der ortsbasierten Erinnerung, mit eigenem Marker. Bei der ortsbasierten Erinnerung gibt es die Möglichkeit, die Karte zu zoomen und auch einen eigenen Marker für die Erinnerung zu setzen. Wird kein eigener Marker hinzugefügt, wird der aktuelle Standort als Erinnerung genutzt. Der eigene Marker gibt dem Benutzer jedoch die Möglichkeit, abweichend von der eigenen Position, den Ort in der Umgebung für die Erinnerung selbst zu bestimmen. Gründe dafür können sein: Die Standortbestimmung ist zu ungenau, oder die Erinnerung wird an einer anderen Position eingerichtet. Die Erinnerungen können durch Auswahl des Icons in der Liste der Startansicht auch geändert oder gelöscht werden.

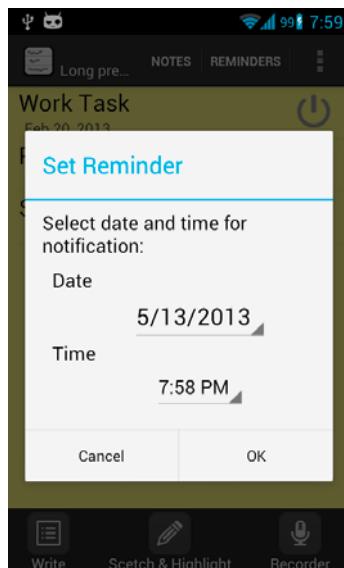


Bild 1.5
Ansicht der zeitbasierten Erinnerung

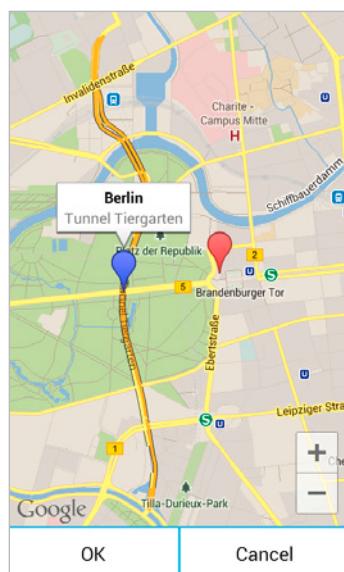
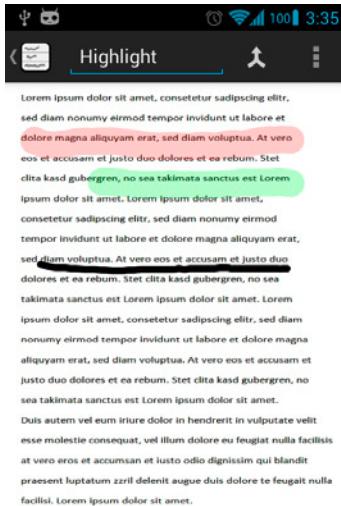


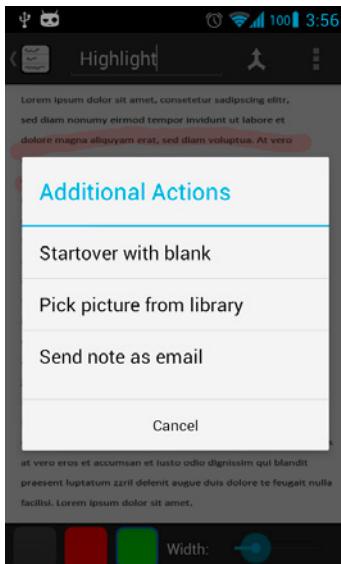
Bild 1.6
Ansicht der ortsbasierten Erinnerung mit eigenem Marker

Der Benutzer hat nicht nur die Möglichkeit, einfache Skizzen als Bild-Notiz anzufertigen, sondern kann auch Fotos aus der Galerie wählen, diese mit Markierungen versehen und abspeichern. Mit den beiden Farben Rot und Grün wird deshalb ausschließlich transparent gezeichnet, wie in Bild 1.7 zu sehen. Außerdem kann die Breite der Markierung über den Schieberegler eingestellt werden.

**Bild 1.7**

Bearbeitungsansicht für Skizzen und Markierungen

Über das OPTIONS-Menü in der oberen ActionBar stehen dem Benutzer weitere Funktionen, je nach Notiz-Typ, zur Verfügung. Im Fall einer Bild-Notiz zum Beispiel, kann mit einem leeren Bild neu gestartet werden, ein Bild aus der Galerie gewählt oder die Notiz per E-Mail versendet werden, wie in Bild 1.8 zu sehen ist.

**Bild 1.8**

Optionen für Skizzen und Markierungen

Da es sich bei der vorgestellten App nicht nur um ein reales Projekt, sondern auch um das Beispiel für dieses Buch handelt, werden wir den Funktionsumfang der App nicht erweitern oder ändern, sodass Sie im Rahmen dieses Buches möglichst eins zu eins genau die App entwickeln, die sich auch für jedermann kostenlos im Google Play Store befindet.



HINWEIS: Wenn wir Feedback oder Bugs von Usern mitgeteilt bekommen, werden wir dies, sofern notwendig, selbstverständlich in eine umgehende Aktualisierung der App einfließen lassen. Als kostenlosen Zusatz zu diesem Buch werden Sie dann eine ausführliche Beschreibung der vorgenommenen Änderungen erhalten. Die PDF wird unter www.downloads.hanser.de zum Download zur Verfügung stehen. Auf dem gleichen Weg werden wir Sie bei Bedarf mit Zusatzinformationen oder Bonuskapiteln versorgen.

■ 1.5 Danksagung

Ein Buch ist keine Einzelleistung. Als Autoren tragen wir viele Bausteine zusammen. Dieses Buch basiert einerseits auf Erfahrungen und Überlegungen aus eigenen Projekten, andererseits aber auch aus öffentlich zugänglichen Quellen des World Wide Web, deren Autoren und Organisatoren wir unseren Dank aussprechen. Ebenso danken wir Frau Sieglinde Schärl und Frau Julia Stepp vom Hanser Verlag für die Anregung zu diesem Buch und die sehr gute Zusammenarbeit.

Besonderen Dank möchte ich, Jochen Baumann, meiner Lebensgefährtin Anke und meinem Sohn Fabian aussprechen, die mit unendlicher Geduld und Unterstützung dazu beitrugen, dass dieses Buch entstehen konnte.

Oberhausen/Essen, im August 2013

Jan Tittel und Jochen Baumann

2

Einrichten der Arbeitsumgebung

In diesem Kapitel wird die Installation und Einrichtung der Entwicklungsumgebung Eclipse und des ADT-Plug-ins unter Windows beschrieben. Weiterhin werden die Android Developer Tools kurz vorgestellt. Bevor wir uns um die Einrichtung von Eclipse kümmern, starten wir mit der Installation von Java.

■ 2.1 Installation von Java

Als Erstes sollten Sie eine aktuelle Version des Java Development Kit (JDK) installieren. Dies kann parallel zur Java Runtime Environment (JRE) erfolgen. Den Download zum JDK (Bild 2.1) finden Sie unter:

- <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- <http://java.oracle.com>



Bild 2.1 Downloadlink zum JDK

Downloaden Sie das passende JDK für Ihr Betriebssystem (Bild 2.2). Sie sollten immer die aktuellste Version auswählen.

The screenshot shows the Java SE Development Kit 7u15 download page. At the top, there is a license acceptance dialog with the text: "You must accept the Oracle Binary Code License Agreement for Java SE to download this software." Below this are two radio buttons: "Accept License Agreement" (selected) and "Decline License Agreement".

Product / File Description	File Size	Download
Linux x86	106.64 MB	jdk-7u15-linux-i586.rpm
Linux x86	92.97 MB	jdk-7u15-linux-i586.tar.gz
Linux x64	104.77 MB	jdk-7u15-linux-x64.rpm
Linux x64	91.68 MB	jdk-7u15-linux-x64.tar.gz
Mac OS X x64	143.75 MB	jdk-7u15-macosx-x64.dmg
Solaris x86 (SVR4 package)	135.52 MB	jdk-7u15-solaris-i586.tar.Z
Solaris x86	91.94 MB	jdk-7u15-solaris-i586.tar.gz
Solaris SPARC (SVR4 package)	135.92 MB	jdk-7u15-solaris-sparc.tar.Z
Solaris SPARC	95.26 MB	jdk-7u15-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	22.92 MB	jdk-7u15-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.59 MB	jdk-7u15-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	22.53 MB	jdk-7u15-solaris-x64.tar.Z
Solaris x64	14.96 MB	jdk-7u15-solaris-x64.tar.gz
Windows x86	88.75 MB	jdk-7u15-windows-i586.exe
Windows x64	90.4 MB	jdk-7u15-windows-x64.exe

Bild 2.2 Versionsauswahl von Java

Installieren Sie nun das JDK und setzen Sie gegebenenfalls die Windows-Path-Variable zum JDK-bin-Ordner wie folgt:

1. Öffnen Sie in der Systemsteuerung den Punkt SYSTEM.
2. Klicken Sie auf ERWEITERTE SYSTEMEINSTELLUNGEN und dann auf UMGEBUNGSVARIABLEN.
3. Wählen Sie aus den *Systemvariablen* den Eintrag Path aus, und rufen Sie anschließend den Befehl BEARBEITEN auf, worauf das Fenster wie in Bild 2.3 angezeigt wird.
4. Tragen Sie in das Feld Wert der Variablen den Pfad zum JDK ein (Bild 2.3).

Zum Abschluss überprüfen Sie die Java-Umgebung, öffnen die Eingabeaufforderung (cmd) und geben java -showversion ein. Sie sollten jetzt die Ausgabe aus Bild 2.4 erhalten.

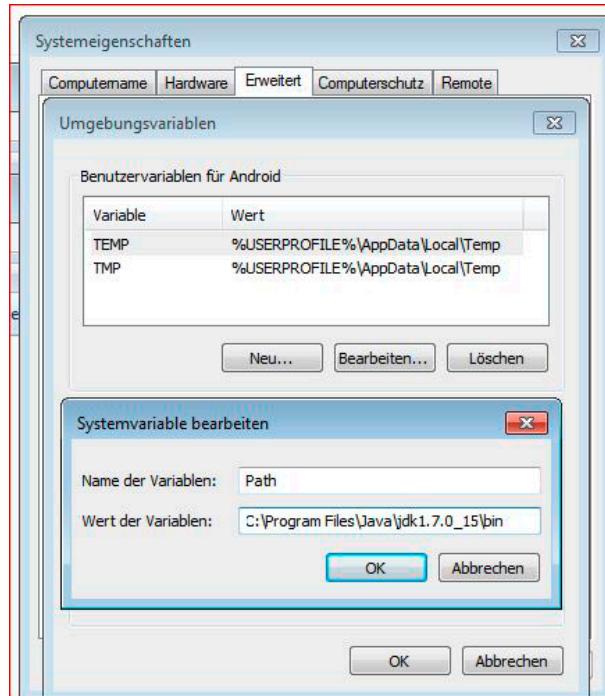


Bild 2.3 Windows-Systemvariablen: Path-Variable JDK

```
java version "1.7.0_15"
Java(TM) SE Runtime Environment (build 1.7.0_15-b03)
Java HotSpot(TM) 64-Bit Server VM (build 23.7-b01, mixed mode)

Verwendung: java [-options] class [args...]
              (zur Ausf"hrung einer Klasse)
      oder    java [-options] -jar jarfile [args...]
              (zur Ausf"hrung einer JAR-Datei)
wobei options Folgendes umfasst:
  -d32          Verwendet ein 32-Bit-Datenmodell, sofern verf"gbart
  -d64          Verwendet ein 64-Bit-Datenmodell, sofern verf"gbart
  -server        zur Auswahl der "server" VM
  -hotspot       ist ein Synonym f"r die "server" VM [verworfen]
                 Die Standard-VM ist server.

  -cp <Class-Suchpfad von Verzeichnissen und .zip-/ .jar-Dateien>
  -classpath <Class-Suchpfad von Verzeichnissen und .zip-/ .jar-Dateien>
```

Bild 2.4 Ausgabe des Befehls Java-showversion



PRAXISTIPP: Die Ausgabe aus Bild 2.4 ist die Ausgabe des JDK auf einem Windows 7 64-bit-System. Unter einem 32-bit-System erscheint nat"rlich eine entsprechend andere Ausgabe.

■ 2.2 Einrichtung und Konfiguration von Eclipse und ADT

Als Nächstes benötigen Sie eine aktuelle Version von Eclipse und dem ADT-Plug-in. Google stellt hier mittlerweile ein komplettes Paket zum Download zur Verfügung. Sie finden das Paket unter <http://developer.android.com>.

Sie können Eclipse und das ADT-Plug-in aber auch einzeln installieren. Eine Anleitung dazu finden Sie unter <http://developer.android.com/sdk/installing/index.html>.

1. Downloaden Sie das Bundle von der Android-Developer-Seite (<http://developer.android.com>).
2. Entpacken Sie die *zip*-Datei auf Ihrer Festplatte.
3. Verschieben oder kopieren Sie den Ordner *Eclipse* zur besseren Übersicht in ein separates Verzeichnis mit einfacher Namensgebung Ihrer Wahl (Bild 2.5).

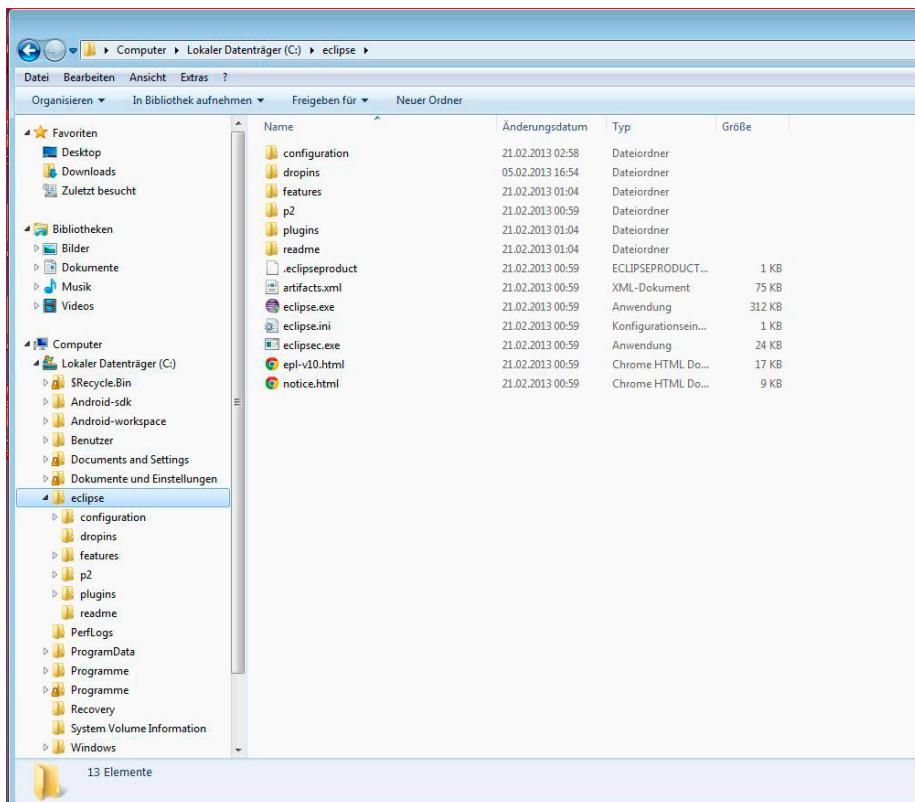


Bild 2.5 Ordner Eclipse

4. Als Nächstes legen Sie ein Verzeichnis für das Android-SDK an und kopieren den Ordner *sdk* und die Datei *SDK Manager.exe* in diesen Ordner (Bild 2.6).

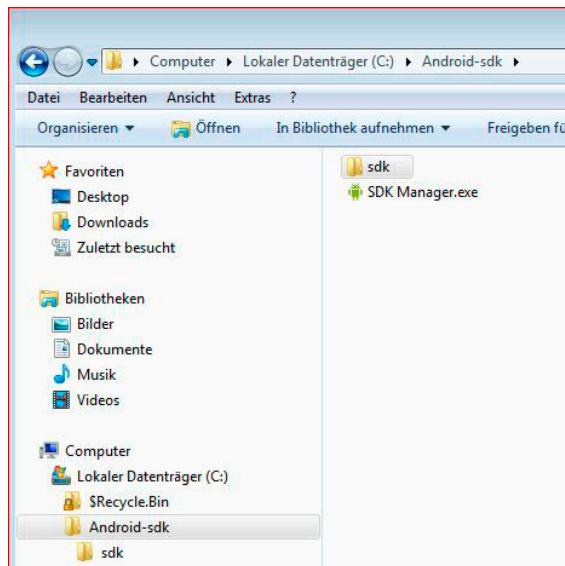


Bild 2.6 Ordner Android-SDK

5. Jetzt tragen Sie, wie bei der Java-Installation, den Pfad zu den Android Plattform-Tools als Systempfad ein, wie in Bild 2.7 abgebildet.

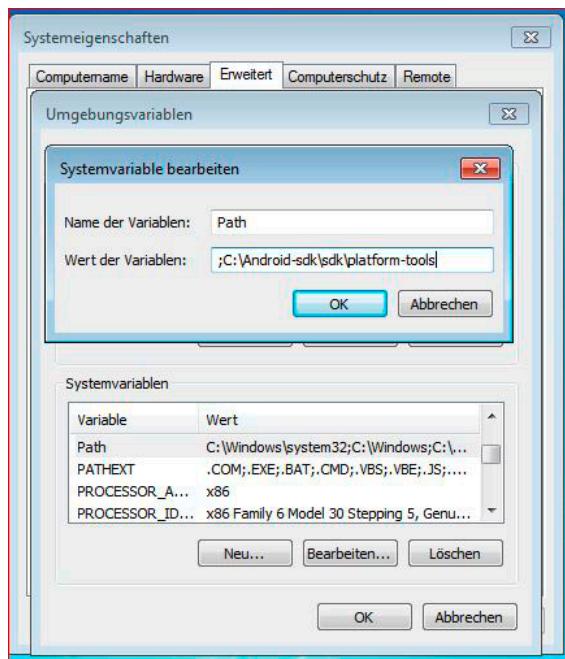


Bild 2.7 Windows-Systemvariablen: Path-Eintrag für die Android-Plattform-Tools

6. Nun aktualisieren Sie das Android-SDK. Dazu starten Sie den SDK-Manager in Ihrem Android-SDK-Ordner durch Doppelklick auf SDK MANAGER.EXE. Auf einigen Systemen erhalten Sie sofort die Fehlermeldung (Bild 2.8), dass der *SDK Manager* keine Verbindung zum Update-Server herstellen konnte. Dies liegt meist an Ihrer Antivirus-/Security-Lösung oder an der Windows-Firewall.

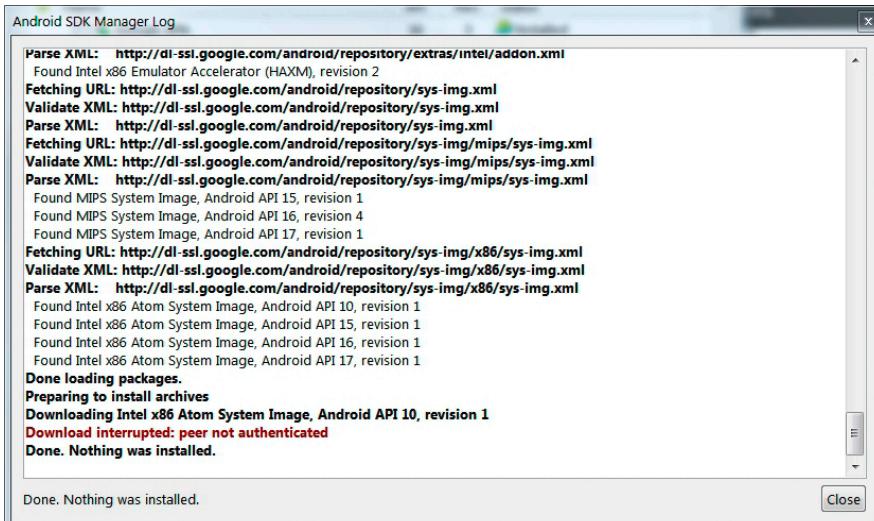


Bild 2.8 Fehlermeldung beim Start von SDK Manager.exe

Sie beheben das Problem, in dem Sie auf *TOOLS/OPTIONS* klicken und im Bereich *Others* (Bild 2.9) die Option *Force https://...sources to be fetched using http://...* aktivieren.

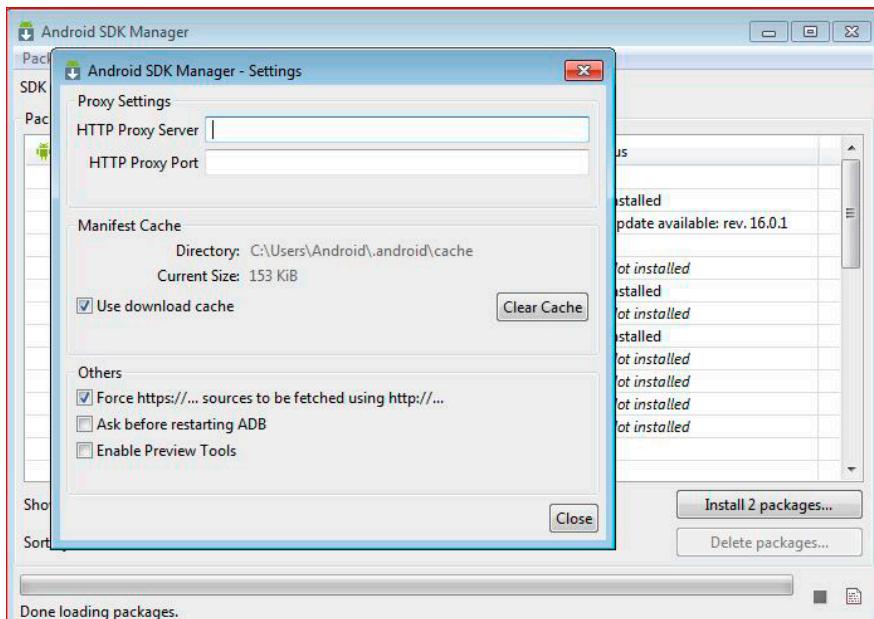


Bild 2.9 Tools/Options im Android-SDK-Manager

7. Ask before restarting ADB fragt nach, bevor die Android Debug Bridge (ADB) neu gestartet wird.
8. Aktivieren Sie die Option Enable Preview Tools, werden Ihnen auch die Vorab-Versionen des SDK angezeigt.



HINWEIS: Diese Versionen enthalten meist noch Fehler. So hatte die Preview Version 21.1r massive Probleme mit Java-64-bit. Sie sollten diese Versionen nicht in einer produktiven Entwicklungsumgebung einsetzen, sondern nur zu Testzwecken.

9. Nun kann das Android-SDK aktualisiert werden. Wenn nötig, installieren Sie auf jeden Fall die Pakete *Google USB-Driver*, *Google APIs* sowie *Samples for SDK* (Bild 2.10).

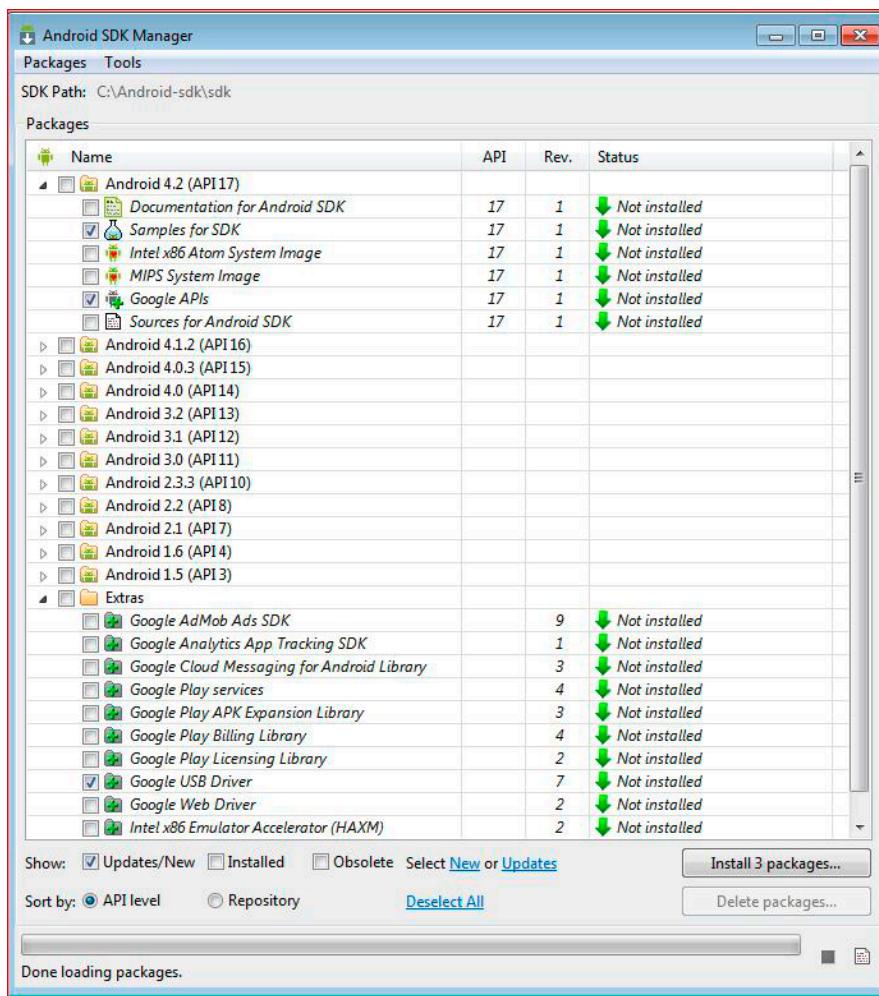


Bild 2.10 Android-SDK-Installation

10. Akzeptieren Sie die Lizenzbedingungen, und klicken Sie auf **INSTALL** (Bild 2.11).

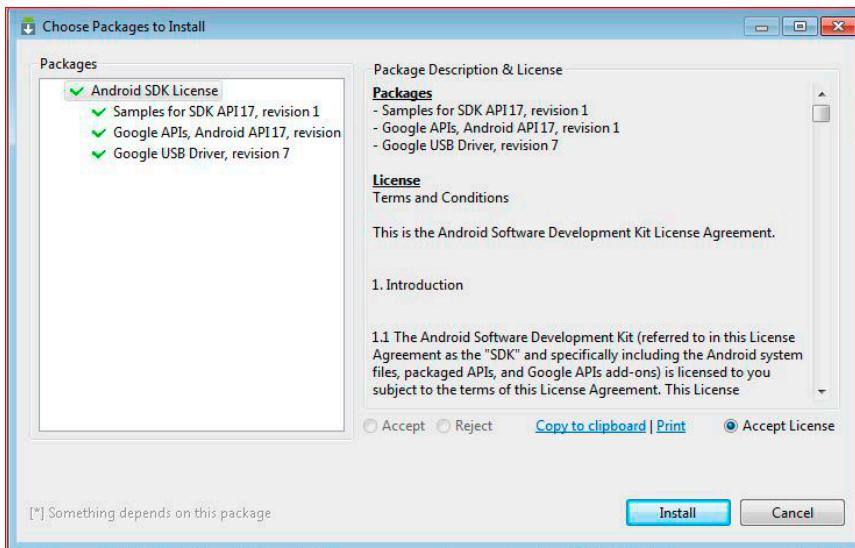


Bild 2.11 Lizenzbedingungen zur Installation

11. Nun können wir Eclipse zum ersten Mal starten. Führen Sie dazu in Ihrem Eclipse Ordner einen Doppelklick auf **ECLIPSE.EXE** aus. Jetzt werden Sie nach einem Workspace gefragt (Bild 2.12). Akzeptieren Sie **nicht** den angegebenen Vorschlag, sondern tragen hier einen eigenen Pfad ein. Benutzen Sie einen separaten Ordner **außerhalb** des Verzeichnisses von Eclipse.

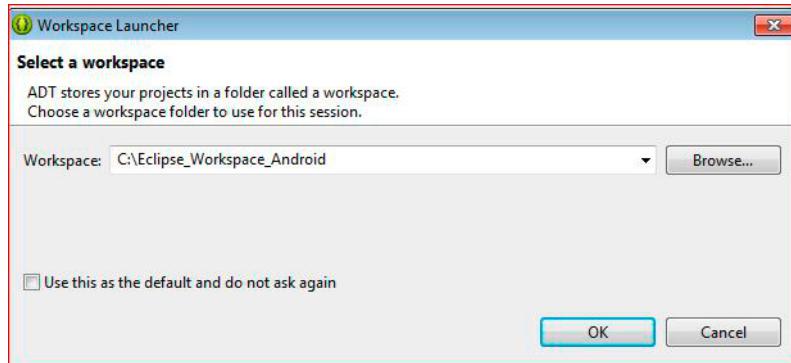


Bild 2.12 Auswahl Eclipse-Workspace

12. Nun sollten Sie eine Fehlermeldung erhalten, dass der Pfad zum Android-SDK nicht gefunden wurde. Klicken Sie auf **OPEN PREFERENCES**, wählen den Ordner Ihres Android-SDK und bestätigen mit **APPLY**. Es werden Ihnen nun die installierten SDK-Versionen angezeigt (Bild 2.13).
13. Sollten Sie die Abfrage nicht erhalten, finden Sie diese Einstellung unter **WINDOW/PREFERENCES/ANDROID**.

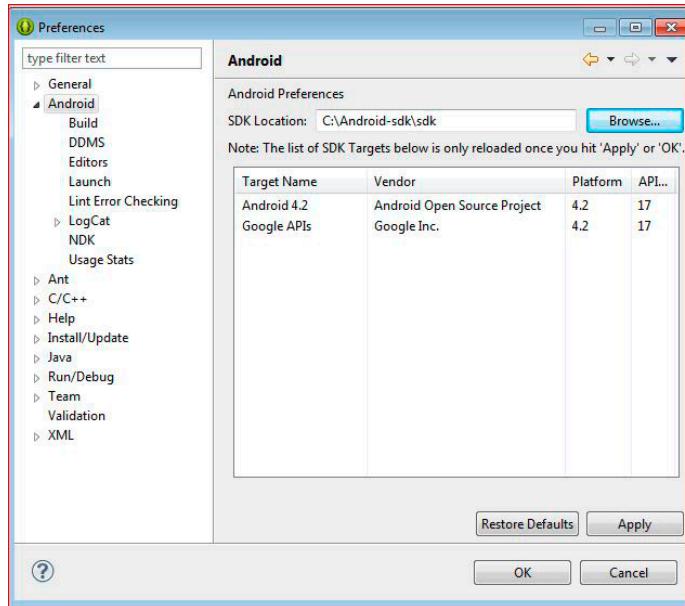


Bild 2.13 Eclipse-Einstellungen: Pfad zum Android-SDK

14. Als Nächstes beheben wir ein kleines Problem der Software-Einstellungen. Leider tritt dieses kleine Ärgernis seit einiger Zeit auf. Unter **WINDOW/PREFERENCES/INSTALL/UPDATES/AVAILABLE SOFTWARE SITES/**, ist der Eintrag für die *URL* des ADT-Plug-ins fehlerhaft. Editieren Sie hier den Eintrag ohne Namen (Bild 2.14) und aktivieren Sie ihn. Den Namen können Sie frei wählen.

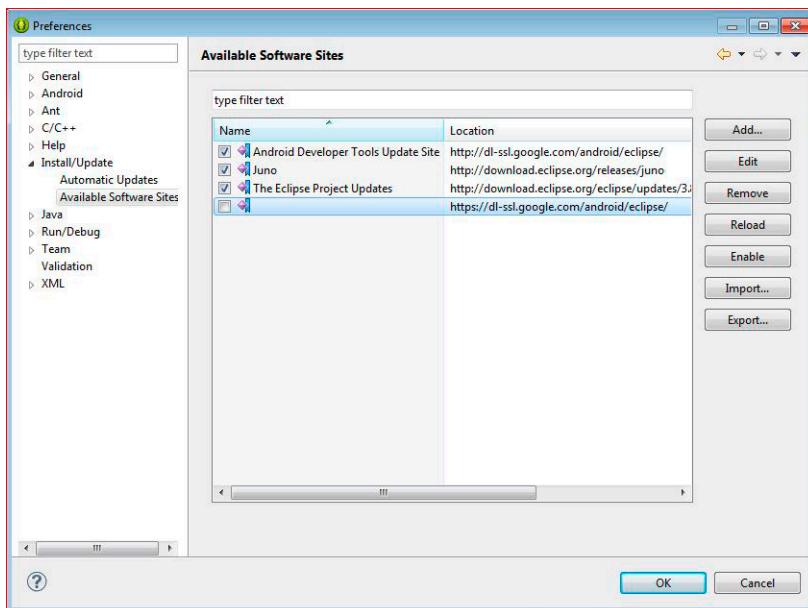


Bild 2.14 Eclipse-Einstellungen: Fehlerhafter Eintrag des Namens der URL

15. Sie sollten immer die SSL-Updates bevorzugen, da diese sicherer sind. Deaktivieren Sie dazu den Eintrag <http://dl-ssl.google.com/android/eclipse/> in der Liste oder entfernen ihn (Bild 2.15).

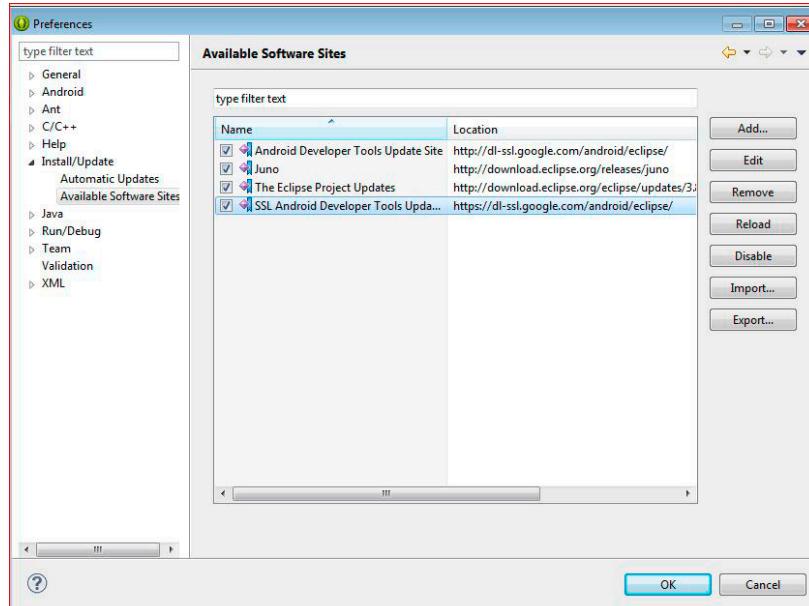
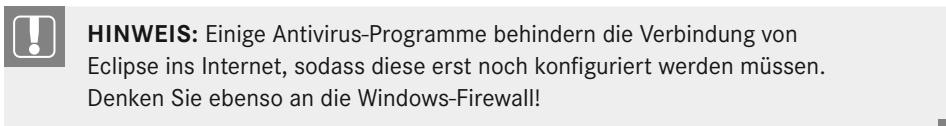


Bild 2.15 Eclipse-Einstellungen: Available Software Sites

16. Meist ist schon ein Java Runtime Environment (JRE) installiert, wie in Bild 2.16 zu sehen. Damit Eclipse fehlerfrei funktioniert, benötigen wir jedoch das JDK. Das JDK bringt, wie Sie schon im Installationspfad gesehen haben, eine eigene Runtime-Umgebung mit.
17. Öffnen Sie dazu die PREFERENCES in Eclipse, die Sie unter WINDOW/PREFERENCES/JAVA/INSTALLED JREs finden (Bild 2.16).
18. Erstellen Sie einen neuen Eintrag mit ADD..., belassen die Auswahl der nächsten Seite auf Standard VM und klicken NEXT. Hier tragen Sie den Pfad zum JDK ein. Eclipse erkennt automatisch die Version, den Namen und den Pfad zum JRE. Jetzt bestätigen Sie mit FINISH (Bild 2.17).

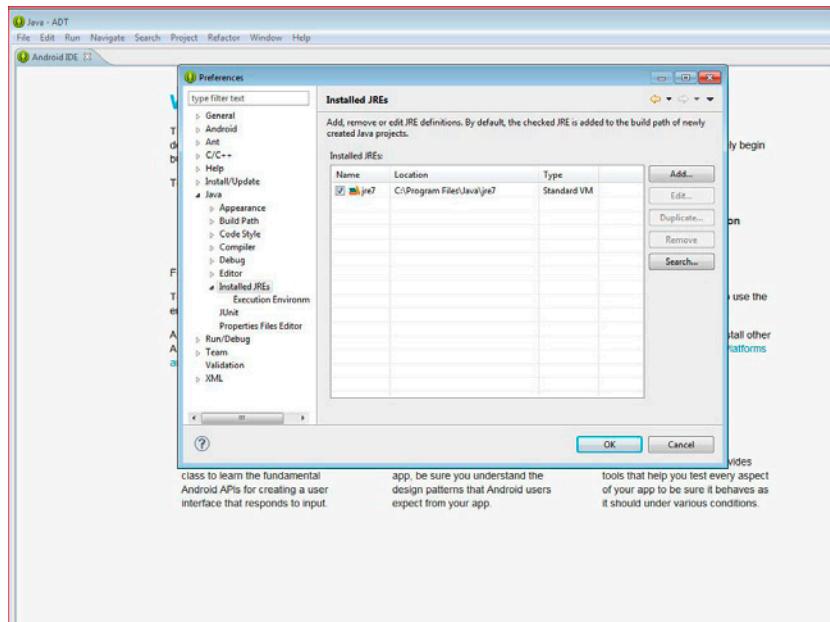


Bild 2.16 Eclipse-Einstellungen: Java Installed JREs

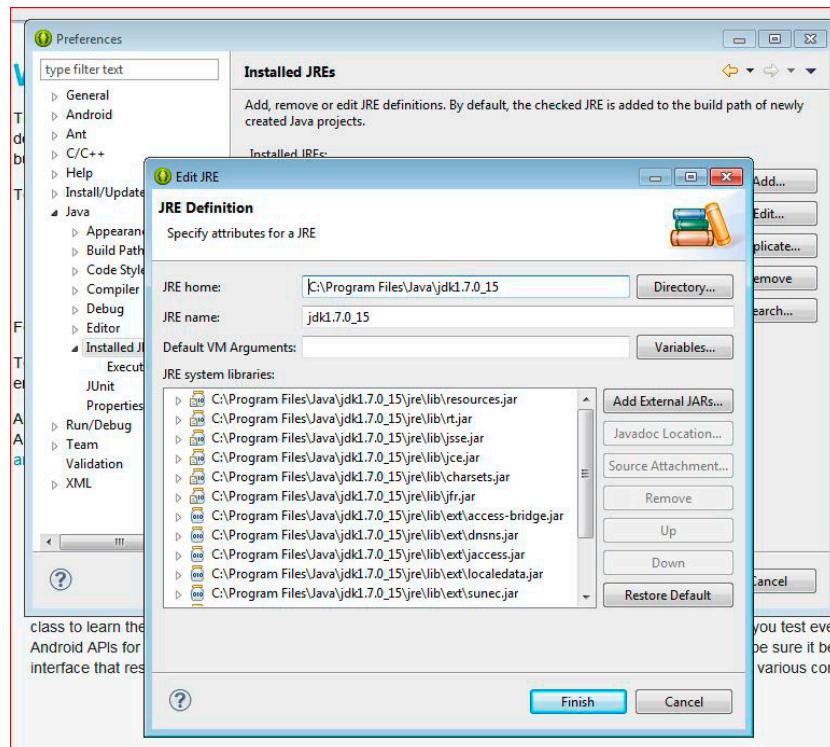


Bild 2.17 Eclipse Installed JRE: Neuer JDK-Eintrag

19. Im Anschluss aktivieren Sie den Eintrag des *JDK* und schließen die Einstellungen. Sie können den Eintrag des *JRE* entfernen. Dieser Eintrag wird nicht benötigt (Bild 2.18).

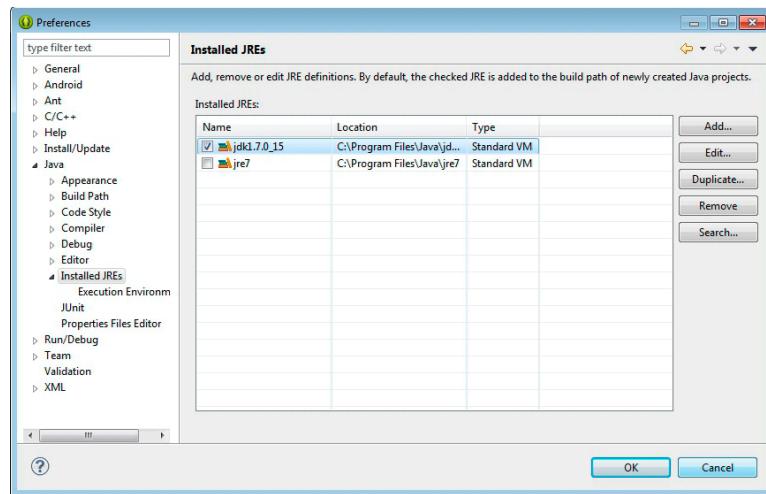


Bild 2.18 Eclipse-Einstellungen: Java-Installed-JREs-Abschluss

20. Zum Abschluss der Vorbereitungen von Eclipse müssen Sie noch den Compliance Level des Compilers überprüfen. Wie auf der Android-Developer-Seite unter SYSTEM REQUIREMENTS angegeben (<http://developer.android.com/sdk/index.html>), benötigt die Android-Plattform das *JDK 6*. Es ist aber das *JDK 7* installiert. Dieses ist abwärtskompatibel, und wir müssen dem Compiler nur mitteilen, dass dieser einen Java-Version 6 (1.6)-konformen Code erstellen soll. Diese Einstellung finden Sie in den PREFERENCES von Eclipse unter JAVA/COMPILER. Der Eintrag *Compiler compliance level:* sollte auf den Wert *1.6* eingestellt sein (Bild 2.19).

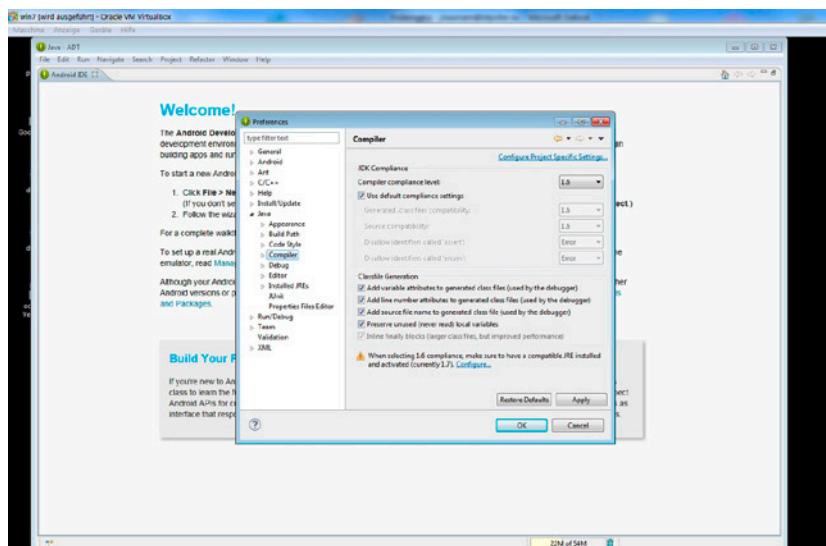


Bild 2.19 Eclipse-Einstellungen: Java Compiler compliance level

■ 2.3 Geräte für die Entwicklung einrichten

Sie müssen die Geräte nicht, wie bei anderen Herstellern, erst auf der Entwicklerseite freischalten und sind auf eine bestimmte Anzahl beschränkt, jedoch muss auch Ihr Smartphone wissen, dass es jetzt für die Entwicklung genutzt wird. Dies ist erforderlich, um die App von Eclipse direkt auf Ihrem Gerät zu installieren und dort zu debuggen.

1. Entwickler, die ein neueres Gerät besitzen, müssen zuerst den Menüpunkt *//Entwickleroptionen* sichtbar machen. Dazu öffnen Sie auf Ihrem Smartphone/Tablet die *Einstellungen*. Öffnen Sie dann den Screen *Über das Tablet* bzw. *Über das Telefon* (Bild 2.20).

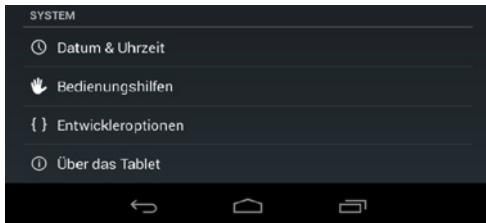


Bild 2.20
Einstellungen Tablet Nexus

2. Im Screen *Über das Tablet* klicken Sie mindestens siebenmal hintereinander auf den Eintrag *Build-Nummer* (Bild 2.21). Sie erhalten eine kurze Meldung, wenn die Entwickleroptionen aktiviert werden.



Bild 2.21
Screen Nexus 7 über das Tablet

3. Jetzt müssen Sie noch die Optionen aktivieren, damit Ihr Gerät im ADT angezeigt wird und Sie Ihre App automatisch von Eclipse installieren und ausführen können. Sie müssen mindestens den Punkt *USB-Debugging* aktivieren. Aktivieren Sie auch den Eintrag *Aktiv lassen*. Damit bleibt das Display aktiv, während Sie Ihr Gerät am Computer angeschlossen haben (Bild 2.22), und Sie müssen es nicht nach kurzer Zeit reaktivieren, da dies während der Entwicklung sehr umständlich ist.
4. Die Option *Apps über USB bestätigen* brauchen Sie nur aktivieren, wenn Sie fremde Apps als *apk-Datei* downloaden und manuell installieren wollen. Diese werden dann einer Prüfung auf Viren und Schad-Software unterzogen. Diese Prüfung erfolgt auf der Basis von Einträgen in einer Liste des Google Play Stores, die nur Einträge umfasst, die bei einer sehr oberflächlichen Prüfung aufgefallen sind.

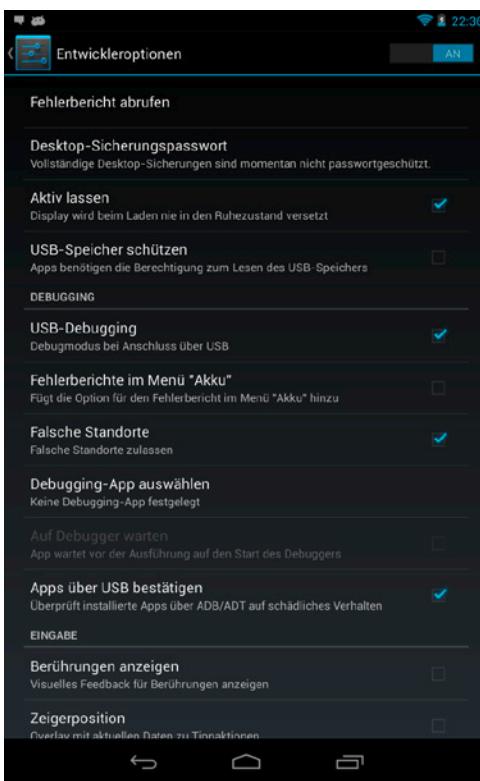


Bild 2.22
Debug-Einstellungen Tablet Nexus 7

5. Zum Abschluss überprüfen Sie, ob Ihr Gerät richtig erkannt wird. Dazu starten Sie Eclipse und öffnen die *DDMS Perspektive* unter *WINDOW/OPEN PERSPECTIVE/DDMS* (Bild 2.23).
6. Innerhalb des *DDMS (Dalvik Debug Monitor)* sollte jetzt Ihr Gerät angezeigt werden (Bild 2.24).

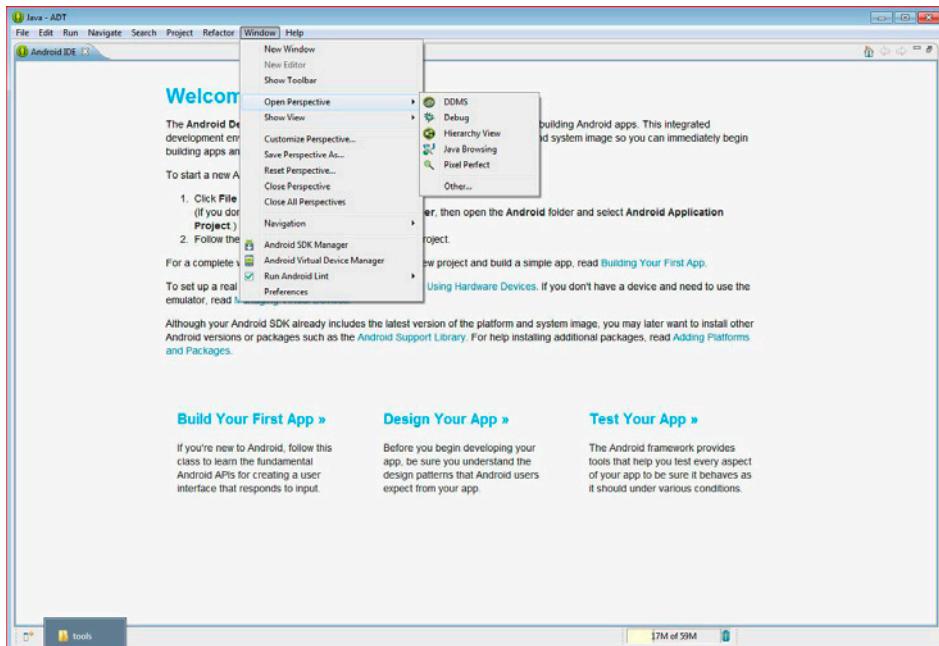


Bild 2.23 Open Perspective Eclipse

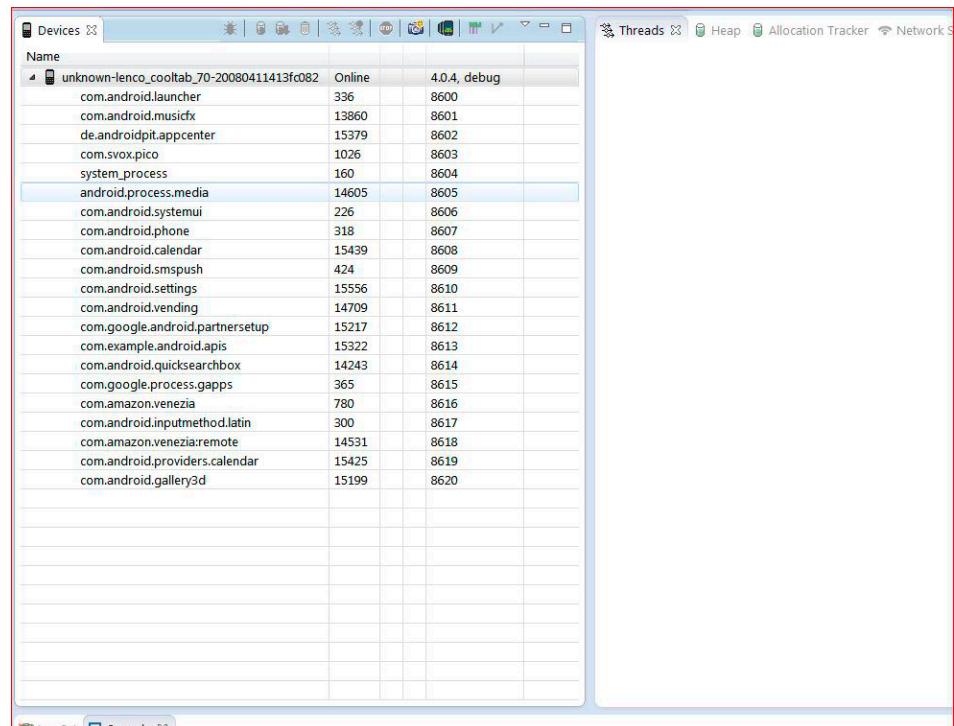


Bild 2.24 Eclipse DDMS Perspective mit angeschlossenem Tablet

3

Schnelleinstieg in Eclipse und Java

In diesem Kapitel erstellen Sie Ihre erste App. Sie erhalten einen praktischen Überblick über die IDE Eclipse (IDE = Integrated Development Environment), legen ein neues Projekt an und erstellen eine einfache App. Diese testen Sie sowohl im Emulator wie auch auf Ihrem Android-Gerät. Zum Abschluss dieses Kapitels erhalten Sie noch einen kurzen Crash-Kurs in Java.



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

■ 3.1 Die erste App mit Eclipse und dem ADT erstellen

In diesem Abschnitt erstellen Sie Ihre erste eigene App. Neben dem grundlegenden Umgang mit Eclipse, erläutern wir auch die Struktur eines Android-Projekts. Anschließend wird die Oberfläche mit grundlegenden Steuerelementen erstellt, und die Steuerelemente werden mit einer einfachen Konfiguration angepasst. Wir beschränken uns hier auf ein Minimum für eine funktionsfähige Anwendung. Schließlich werden Sie einer Schaltfläche eine Methode hinzufügen, die auf das Tippen des Anwenders auf den Bildschirm reagiert – nein nicht, wie so oft, „Hello World“.



HINWEIS: Wir verwenden in diesem sowie in den folgenden Kapiteln die englischen Begriffe der Entwicklungsumgebung. Es ist zwar möglich, Eclipse zu lokalisieren, jedoch setzen wir die Downloadversion von Google mit dem ADT ein und verzichten auf eine deutsche Lokalisierung.

3.1.1 Ein neues Projekt in Eclipse anlegen

- Starten Sie Eclipse und legen ein neues Projekt an: Klicken Sie dazu in der Menüleiste auf FILE NEW und dann auf ANDROID APPLICATION PROJECT, so wie es auch auf der Start View in Bild 3.1 von Eclipse beschrieben ist.

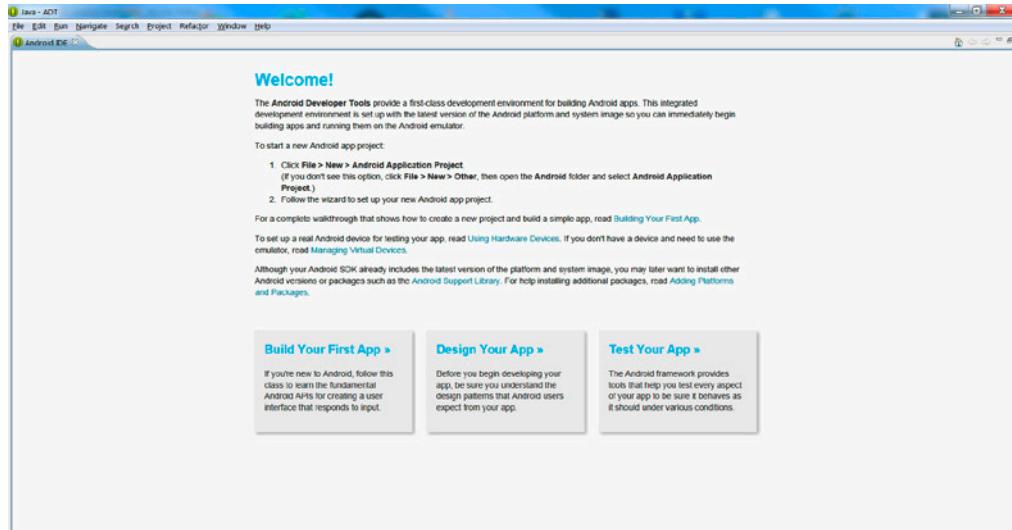


Bild 3.1 Eclipse-Start-Perspektive

- Sollten Sie unter FILE NEW die Projektauswahl das *Android Application Project* nicht finden, so klicken Sie auf FILE NEW OTHERS und wählen hier wie in Bild 3.2 *Android Application Project*.

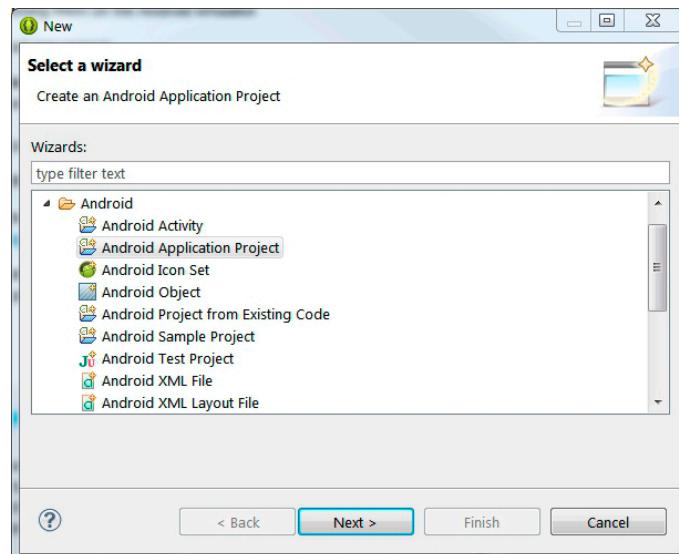


Bild 3.2 Eclipse-Projektassistenten-Auswahl

3. Nun startet der Android-Projektassistent, und Sie werden automatisch durch das Anlegen eines neuen Android-Projekts begleitet.
4. Als Erstes legen Sie den Namen Ihrer App fest (siehe Bild 3.3). Dieser wird so auch im Google Play Store und in Ihrer App angezeigt. Aber haben Sie keine Bedenken, diesen Namen können Sie später noch ändern. Er wird an dieser Stelle nicht in Stein gemeißelt, jedoch sollten Sie diesen nach der Veröffentlichung nicht mehr ändern. Der *Projektname* ist nur für Eclipse interessant und muss eindeutig im Workspace vorhanden sein, das heißt, dieser Name kann in einem anderen Workspace durchaus noch einmal vorkommen, Sie sollten den Projektnamen aber hier nicht ändern. Sie können die Projekte auf Dateiebene anhand der Ordnerstruktur wiedererkennen. Dem Package-Namen kommt wieder eine höhere Bedeutung zu. Über den Paket-Namen wird Ihre App eindeutig identifiziert. Während des gesamten Lebenszyklus, also nach Veröffentlichung, sollten Sie den Paket-Namen nicht ändern. Als Standard bei der Namensgebung hat sich die umgekehrte Notation der Domain der Organisation, wie *eu.scyte.* und einem zusätzlichen Identifier für die Anwendung, durchgesetzt. Die Auswahl des minimalen SDK und des Targets ist von Ihrer Zielsetzung und Programmierung abhängig.

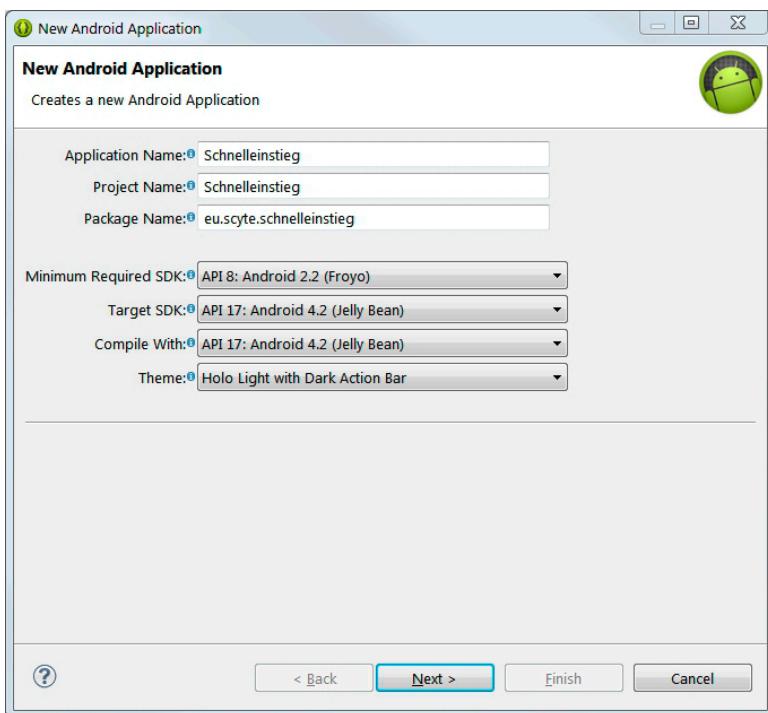


Bild 3.3 New Android Application

5. Mit der Einstellung **Minimum Required SDK** geben Sie den kleinsten notwendigen API-Level an, mit dem Ihre App lauffähig ist. **Target SDK** ist der API-Level, mit dem Ihre App erstellt und getestet wurde. Außerdem sorgt diese Angabe auf Geräten mit einer höheren Android-Version dafür, dass das System die App in einem Kompatibilitätsmodus ausführt und die gewünschte Funktionalität erhalten bleibt. Geben Sie bei *Target SDK* kei-

nen Wert ein, so wird automatisch die Einstellung *Minimum Required SDK* als *Target SDK* gewählt.

Die Wahl des richtigen API-Levels ist abhängig von den Funktionen, die Sie in Ihrer Anwendung nutzen und anbieten wollen, je kleiner der API-Level, desto mehr Geräte können Ihre App nutzen.



PRAXISTIPP: Sie sollten Ihre App derzeit mindestens mit dem API-Level 14 als Target SDK, also Ice Cream Sandwich, erstellen, um von den neuen Oberflächenelementen der Standardversion zu profitieren. Es werden spezielle Support-Bibliotheken bereitgestellt, um frühere Versionen zu unterstützen.

6. **Projektkonfiguration:** Auf dieser Konfigurationsseite (Bild 3.4) geben Sie zuerst an, ob ein individuelles Start-Icon und eine Activity erstellt werden sollen.

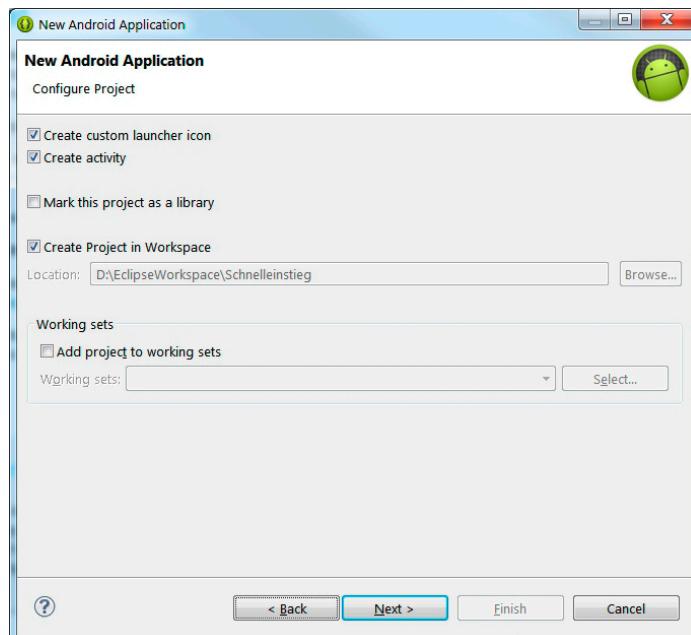


Bild 3.4 Projektkonfiguration für Start-Icon und Activity

7. **Start-Icon erstellen:** Auf dieser Seite des Assistenten (Bild 3.5) können Sie sich ganz einfach ein Launcher Icon für Ihre App erstellen lassen, probieren Sie ein wenig aus.
8. **Activity erstellen:** Hier entscheiden Sie, dass eine Activity und ein Layout erstellt werden sollen (siehe Bild 3.6). Activities sind innerhalb einer Android-Anwendung Bausteine, die meist die Benutzeroberfläche und Interaktionen repräsentieren. Jede Android-Anwendung besteht deshalb aus mindestens einer Activity. Activities können andere Activities aufrufen und mit ihnen Daten austauschen. Jede Activity besteht aus einer in XML definierten Layout-Datei und einer dazugehörigen Java-Klassendatei.

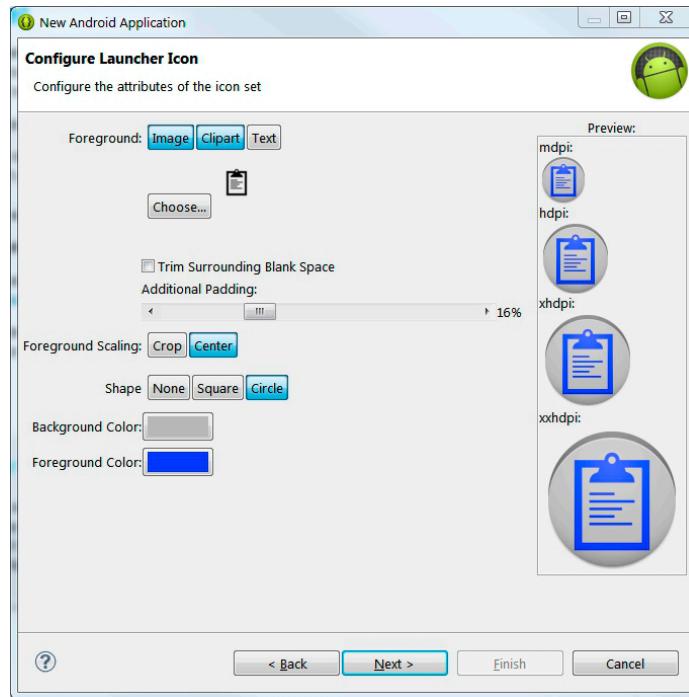


Bild 3.5 Launcher Icon erstellen

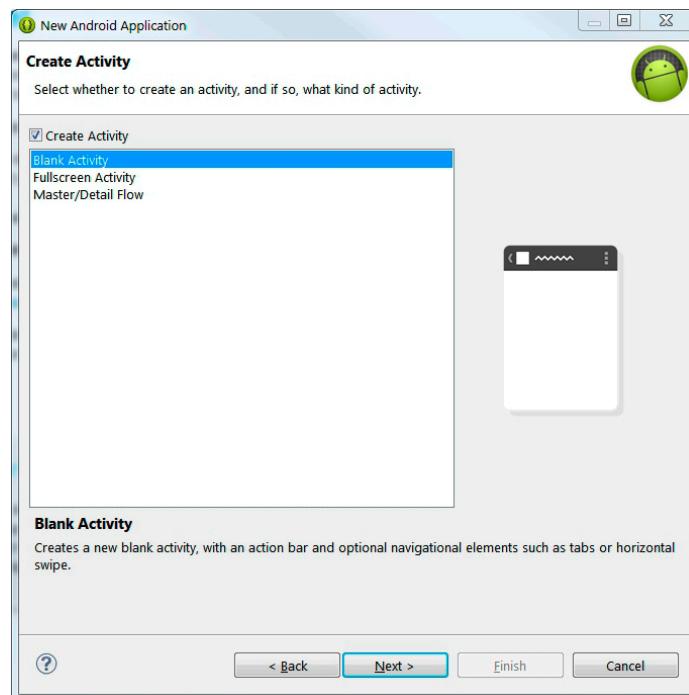


Bild 3.6 Activity erstellen

9. **Activity-Einstellungen:** Auf dieser Seite des Assistenten vergeben Sie die Namen für die Activity und die Layout-Datei. Diese Activity ist der oberste Startpunkt Ihrer Android-Anwendung.

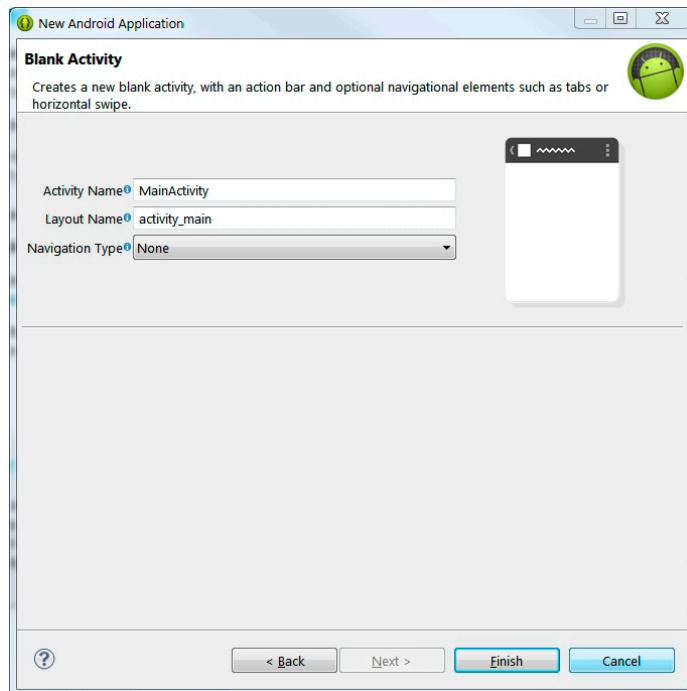


Bild 3.7 Einstellungen der Activity

10. Durch Klicken auf FINISH beenden Sie den Assistenten und haben somit Ihr erstes Android-Projekt erstellt. Sie sollten jetzt wieder die Startansicht wie in Bild 3.1 sehen. Schließen Sie diese Welcome View durch einen Klick auf das X im Karteireiter neben *Android IDE*. Wenn alles richtig funktioniert hat, sehen Sie die Standardprojektansicht.

3.1.2 Erster Einstieg in Eclipse mit ADT

Bevor wir nun unsere erste App anpassen, machen wir uns mit Eclipse ein wenig vertraut. Die Oberfläche von Eclipse ist in Sichten (Views) und Perspektiven (Perspectives) unterteilt. Eine Perspektive ist im Wesentlichen eine Zusammenstellung von einer oder mehreren Sichten für eine bestimmte Anforderung. Perspektiven können individuell angepasst werden. Wir beschränken uns jedoch auf die vorgefertigten Perspektiven und öffnen bei Bedarf weitere Views (Sichten). Eclipse liefert eine Vielzahl von vorgefertigten Perspektiven mit, von denen Sie im weiteren Verlauf einige kennen lernen werden. Die von uns genutzte Standardperspektive zur Erstellung von Android-Anwendungen ist die Java-Perspektive. Views sind in Eclipse einzelne Teilfenster, die beliebig angezeigt, verschoben oder ausgeblendet werden können.

Eine Perspektive oder View öffnen Sie durch einen Klick auf WINDOW/OPEN PERSPECTIVE bzw. SHOW VIEW.

Eclipse-Aufbau kurz erklärt: Die *JAVA-Perspective* (Bild 3.8) von Eclipse mit dem ADT ist in vier Bereiche gegliedert, die eine oder mehrere Views enthalten. Auf der linken Seite sehen Sie den *Package Explorer*. Diesen können Sie sich als eine logische Sicht auf die Projektdaten vorstellen. Es werden alle für die Programmierung notwendigen Daten angezeigt. Der Package Explorer hat keinen Bezug zur physikalischen Ansicht der Daten. Wenn Sie sehen möchten, wie Ihre Daten auf der Festplatte gespeichert werden, müssen Sie die View *Navigator* öffnen, in dem Sie auf WINDOW/SHOW VIEW/NAVIGATOR klicken. Im oberen Mittelteil sehen Sie die Editoren-View. In Bild 3.8 sehen Sie den XML-Layout-Editor des ADT, der eine grafische und textbasierte (XML) Oberfläche bietet. Die Text-Ansicht erreichen Sie, in dem Sie im unteren Bereich des Editors auf ACTIVITY_MAIN.XML klicken. In dieser View werden alle Quelltextdateien, die Sie durch Doppelklick im Package Explorer öffnen, angezeigt. Im rechten Bereich finden Sie die *Outline View* und die *Properties View*. Die Outline View stellt hierarchisch strukturiert die geöffneten Quelltextdateien dar. Sie können mit der Outline View sehr schnell durch Ihren Quelltext navigieren. Gerade bei längerem Quelltext ist dies sehr nützlich. Die View Properties bietet die Möglichkeit, Eigenschaften von markierten View-Elementen anzuzeigen und zu editieren. Im unteren Bereich Mitte/Links sind mehrere Views verankert. Die Konsole zeigt Ihnen die direkte Ausgabe von Meldungen der *ADB* (*Android Debug Bridge*).

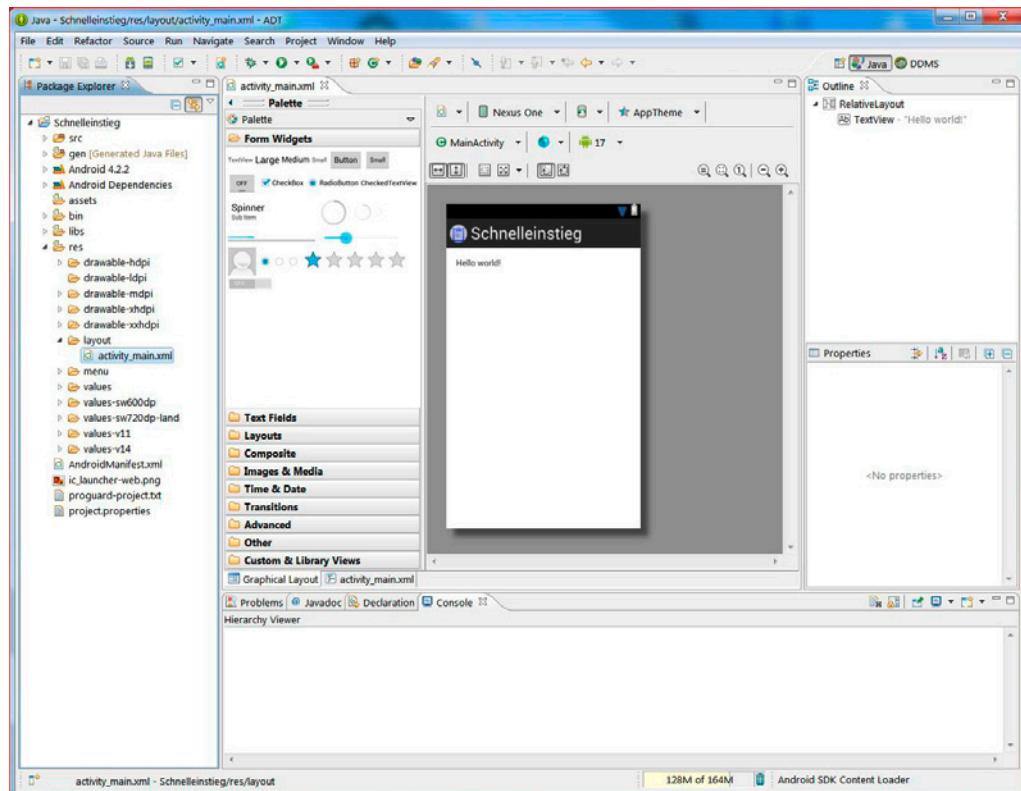


Bild 3.8 Eclipse Java Perspective

Machen Sie sich mit Eclipse und dem ADT ein wenig vertraut. Öffnen Sie einzelne Views und Perspektiven. Haben Sie eine Perspektive angepasst und möchten diese wieder in den Standard zurücksetzen, erreichen Sie dies durch einen Klick auf **WINDOW/RESET PERSPECTIVE**. Wollen Sie Ihre angepasste Perspektive abspeichern, können Sie das durch einen Klick auf **WINDOW/SAVE PERSPECTIVE** erreichen.

Eclipse-ADT-Einstellungen: Sie haben Eclipse nun kennen gelernt. Wir bevorzugen die Anzeige der Zeilennummern im Quellcode. Öffnen Sie dazu die Eclipse-Einstellungen (*Eclipse Preferences*) (Bild 3.9), klicken Sie auf **WINDOW/PREFERENCES**, und navigieren Sie zu *Text Editors* unterhalb *Editors* im Bereich *General*. Aktivieren Sie nun *Show line numbers*.

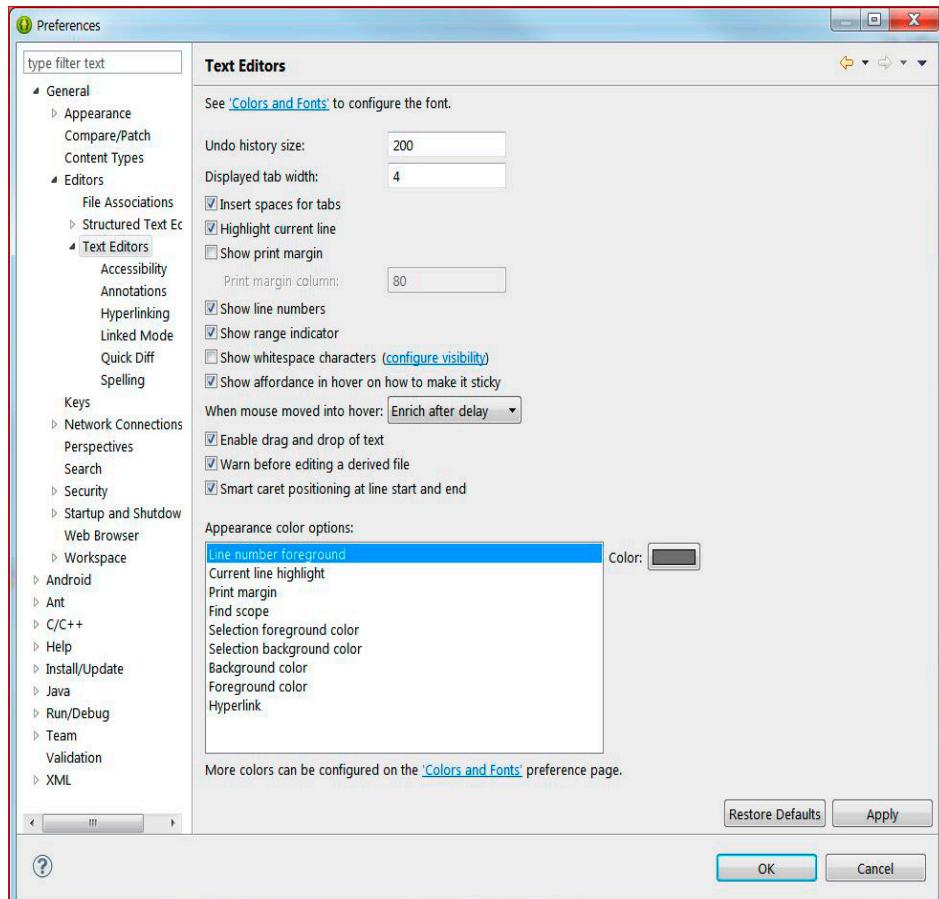


Bild 3.9 Eclipse Preferences

In dem Eclipse-Bereich *Preferences* können Sie die IDE nach Ihren Bedürfnissen einstellen. Warum weisen wir noch einmal auf die Einstellungen von Eclipse hin? In seltenen Fällen kommt es vor, dass Probleme mit Eclipse auftreten können. Doch dann sollten Sie schon ein wenig mit den Preferences vertraut sein.

3.1.3 Die Oberfläche der App anpassen (XML-Layout)

Sie haben ja bereits das Android-Projekt angelegt. Jetzt soll die Oberfläche der App angepasst werden.

Was soll unsere kleine Anwendung überhaupt machen? Unsere App soll Kilometer in Meilen umrechnen. Der User gibt in ein Feld die Kilometer ein, die in Meilen umgerechnet werden sollen. Nach Betätigen des Buttons *berechnen*, wird das Ergebnis angezeigt. Die Berechnung soll auch durch Betätigen des Menüeintrages *berechnen* der *ActionBar* ausgeführt werden.

Was wird benötigt, damit der User ohne Anleitung mit der App umgehen kann?

- Ein Hinweistext
 - Ein Eingabefeld
 - Ein Button oder Aktionsfeld, damit die Berechnung startet
1. Falls die Layout-Datei *activity_main.xml* noch nicht geöffnet ist, öffnen Sie diese. Sie finden Sie im Ordner *res/layout*.
 2. Nun vergeben Sie zuerst eine *Id* für das schon vorhandene *TextView*-Steuerelement. Dies können Sie durch RECHTSKLIICK auf das Element und ASSIGN ID bzw. in der *Properties View* auf der rechten Seite erledigen. Indem Sie den Eintrag *Id* editieren, vergeben Sie folgenden Namen: *textView_hinweisText* (Bild 3.10).

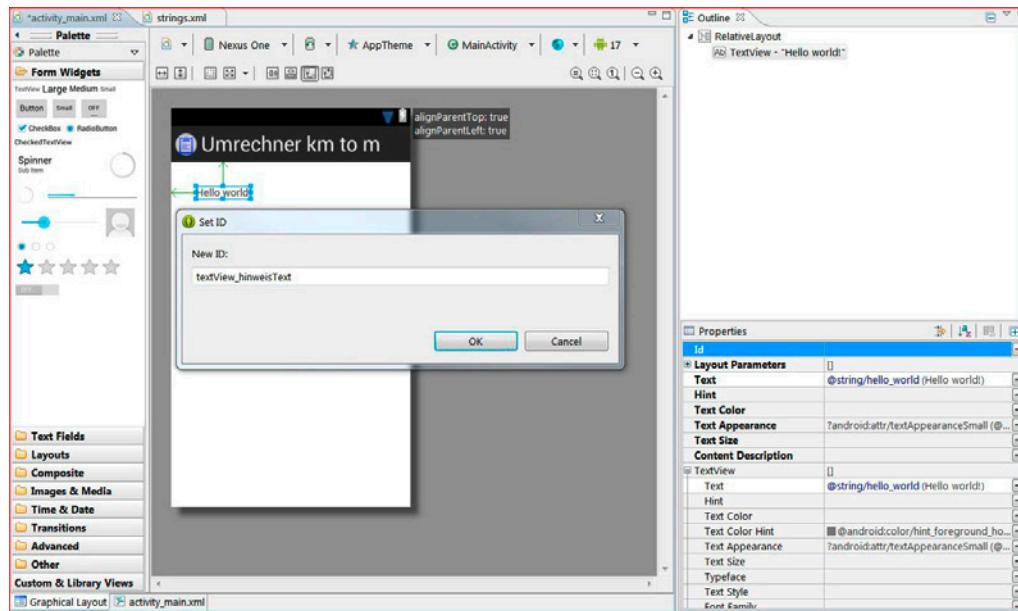


Bild 3.10 Id vergeben

3. Als Nächstes öffnen Sie die Datei *strings.xml* und löschen den Eintrag mit Namen Hello World. Erstellen Sie einen neuen Eintrag mit Namen *textView_hinweis* und folgendem Wert: Hinweis:\n Bitte geben Sie die Kilometer ein,\n die in Meilen umgerech

net werden sollen. Führen Sie diese Änderungen in der XML-Ansicht durch. Ändern Sie außerdem den Wert des Eintrages `app_name` in *Umrechner km to m*. Speichern Sie die Datei `strings.xml`. Dadurch werden alle Verweise auf die Strings aktualisiert. Weisen Sie nun der Textanzeige (*TextView*) den neuen Wert `textView_hinweis` zu. Dazu klicken Sie auf die drei Punkte in der *Properties View* und wählen den Wert aus der Ressource `strings.xml`, wie in Bild 3.11 zu sehen. Achten Sie darauf, dass bei der Auswahl des Strings als Ressource *Project Resources* ausgewählt ist.

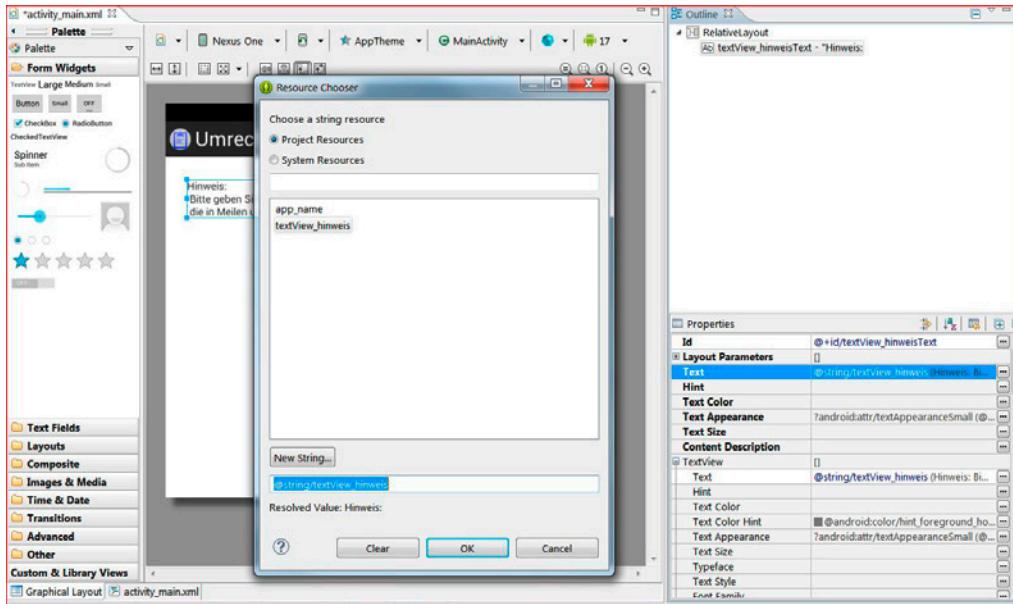


Bild 3.11 *TextView* neuen Text zuweisen

4. Wählen Sie nun auf der linken Seite des grafischen Editors *Text Fields* aus und platzieren das *EditText*-Steuerelement *Plain Text* auf Ihrer Oberfläche, zu erkennen an *abc*. Das ADT-Plug-in zeigt Ihnen den Typen an, wenn Sie mit der Mouse darüberfahren. Ihre Oberfläche sollte nun wie in Bild 3.12 aussehen.
5. Nun benötigen wir noch die Anzeige für die Einheit der Eingabe. Wechseln Sie dazu in der *Palette* auf der linken Seite wieder auf *Form Widgets* und wählen hierfür das Anzeigesteuerelement *TextView* (Small) aus. Platzieren Sie dieses neben der Eingabe. Der grafische Editor unterstützt sie bei der Ausrichtung. Für die Eingabeeinheit erstellen Sie eine *String Resource*, klicken Sie auf die DREI Punkte der Zeile *Text* im Properties-Fenster (Bild 3.11). Im folgenden Fenster *Resource Chooser* klicken Sie auf *NEW STRING* und legen einen neuen Eintrag für den Text dieser *TextView* an (Bild 3.13). Im Feld *New R.string.* den Bezeichner für diesen Text, *label_einheit_km*.

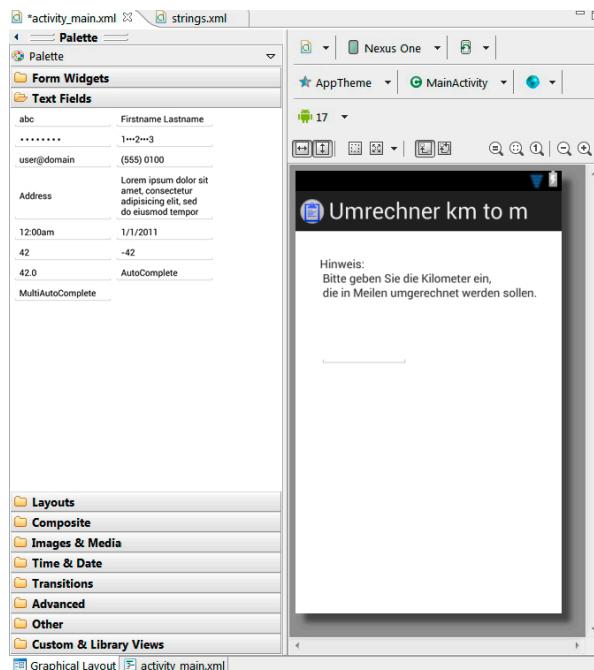


Bild 3.12 Texteingabe auf der Oberfläche der App

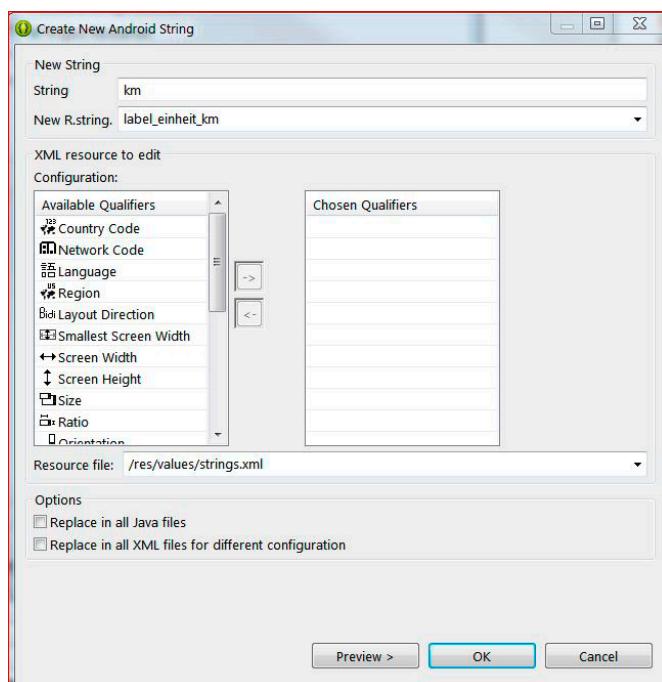


Bild 3.13 Neue String Resource per Assistent

6. Jetzt benötigen Sie noch eine Text-Anzeige für die berechneten Meilen und eine Text-Anzeige für die Einheit. Wählen Sie nun wieder das *TextView*-Steuerelement aus *Form Widgets* (Medium) aus, positionieren es unter der Eingabe und vergeben die *Id*: *textView_ausgabe_miles*. Sie können auch direkt in das Feld *Id* des Properties-Fensters die *Id* eintragen, jedoch müssen Sie dann Ihrer *Id* folgenden Parameter voranstellen: `@+id/`. Der Eintrag muss nun so aussehen: `@+id/textView_hinweisText`. Tragen Sie nun noch *0.00* in die Zeile *Text* der Properties ein. Für die Anzeige der Einheit Meilen positionieren Sie nun wieder *TextView* (Small) neben der Ausgabe *TextView* und tragen in den *Properties* der Zeile *Text* einfach Miles ein.
7. Zum Abschluss fügen Sie noch einen Button hinzu. Wählen Sie das Element *Button* aus *Form Widgets* aus und positionieren diesen unterhalb der Ausgabe *TextView*. Legen Sie als Text der Properties wieder eine neue *String Resource* durch Klick auf die drei Punkte an und tragen als Text *String berechnen* und als Bezeichner *New R.string. berechnen* ein. Unter *View* bei *On Click* geben Sie *onClick* ein. Dieser Eintrag bei *On Click* bewirkt, dass im Code durch einen Klick auf den Button reagiert werden kann. Falls Sie nicht in der Zwischenzeit gespeichert haben, sollten Sie dies jetzt nachholen.
8. Die Oberfläche ist nun so weit fertig gestellt. Sie benötigen nur noch den Eintrag *berechnen* oben in der Menüleiste (*ActionBar*). Dazu öffnen Sie im Ordner *RES/MENU* die Datei *main.xml*, fügen durch *Add...* einen neuen Menüeintrag hinzu, wählen wie in Bild 3.14 *Item* aus und bestätigen mit *OK*. Als *Id* vergeben Sie *action_berechnen*, und als *Title* wählen Sie die schon beim Button angelegte *String Resource* *berechnen*. Bei *Show as action* wählen Sie *always*. Entfernen Sie den Menüeintrag *action_settings* durch *REMOVE*.

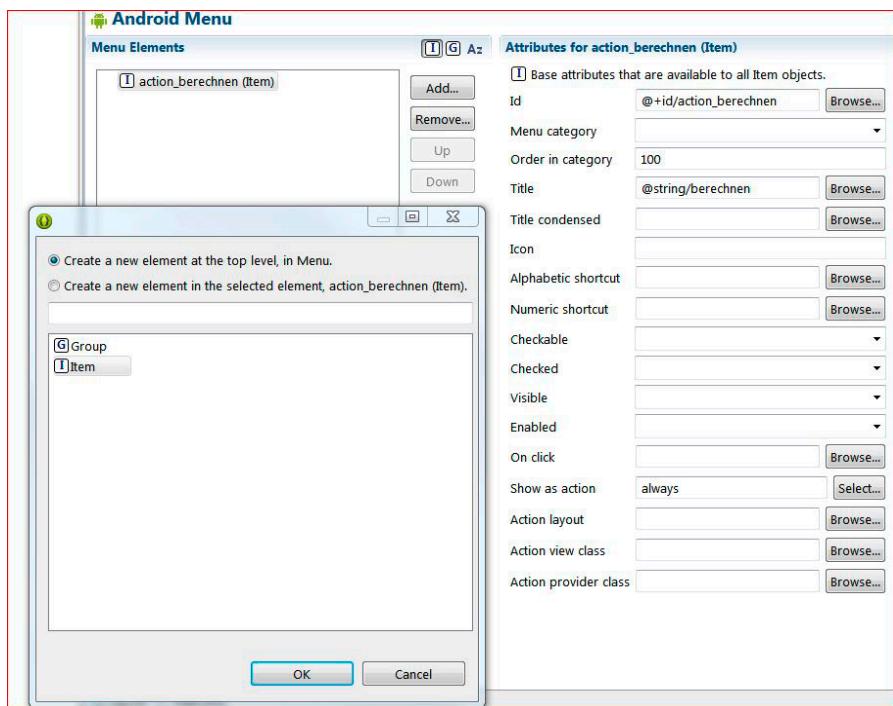


Bild 3.14 Menü main.xml

Die Oberfläche ist nun fertig gestellt und sollte ähnlich dem Bild 3.15 aussehen. Den Menüeintrag *berechnen* sehen Sie jetzt noch nicht, dieser wird erst zur Laufzeit des Programms erzeugt.

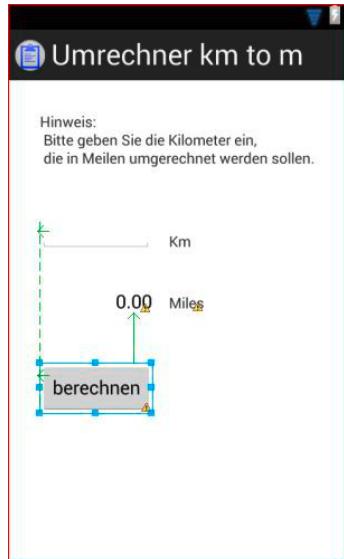


Bild 3.15
Layout der Testanwendung

In den folgenden Listings haben wir zuerst das XML-Layout, dann die Einträge für die Texte und zum Schluss das Menü aufgeführt.

Listing 3.1 XML-Layout: /res/layout/activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin"  
    tools:context=".MainActivity" >  
  
    <TextView  
        android:id="@+id/textView_hinweisText"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_alignParentTop="true"  
        android:layout_centerHorizontal="true"  
        android:layout_marginTop="15dp"  
        android:text="@string/textView_hinweis" />  
  
    <EditText  
        android:id="@+id/editText_eingabe"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_alignLeft="@+id/textView_hinweisText"
```

```

        android:layout_below="@+id/textView_hinweisText"
        android:layout_marginTop="43dp"
        android:ems="5"
        android:inputType="numberDecimal|none" />

    <TextView
        android:id="@+id/textView_label_eingabe"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/editText_eingabe"
        android:layout_marginLeft="16dp"
        android:layout_toRightOf="@+id/editText_eingabe"
        android:text="@string/label_einheit_km" />

    <TextView
        android:id="@+id/textView_ausgabe_miles"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText_eingabe"
        android:layout_marginTop="38dp"
        android:layout_toLeftOf="@+id/textView_label_eingabe"
        android:text="0.00"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/textView_label_miles"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/textView_ausgabe_miles"
        android:layout_alignBottom="@+id/textView_label_miles"
        android:layout_alignLeft="@+id/textView_label_eingabe"
        android:text="Miles" />

    <Button
        android:id="@+id/button_berechnen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText_eingabe"
        android:layout_below="@+id/textView_ausgabe_miles"
        android:layout_marginTop="47dp"
        android:onClick="onClick"
        android:text="berechnen" />

</RelativeLayout>

```

Listing 3.2 Resource Strings: /res/values/strings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Umrechner km to m</string>
    <string name="textView_hinweis">Hinweis:\n Bitte geben Sie die Kilometer ein,\n
die in Meilen umgerechnet werden sollen.</string>
    <string name="label_einheit_km">Km</string>
    <string name="berechnen">berechnen</string>

</resources>

```

Listing 3.3 Resource Menü: /res/menu/menu.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/action_berechnen"
        android:orderInCategory="100"
        android:showAsAction="always"
        android:title="@string/berechnen"/>
</menu>
```

4.1.4 Der App Funktionalität geben

In Abschnitt 3.1.3 haben Sie die Oberfläche der App erstellt, jedoch verfügt die Anwendung über keinerlei Funktionalität. Das werden wir nun nachholen. Dazu müssen Sie zunächst die Datei *MainActivity.java* öffnen. Diese finden Sie im Ordner *src NameIhresPackage.Schnelleinstieg*. Wie Sie schon an der Dateiendung *.java* erkennen können, handelt es sich hierbei um eine Java-Quellcodedatei. In Java-Quellcodedateien wird der Programmcode für Android-Anwendungen abgelegt. In größeren Projekten haben Sie natürlich mehrere dieser Java-Quellcodedateien. In Abschnitt 3.1.5 erläutern wir die Struktur von Android-Projekten.

Doch nun wollen wir unserer App erst einmal Funktionalität verleihen:

1. Als Erstes legen wir den Faktor für die Umrechnung von Kilometern in Meilen fest. Dies erfolgt durch die Festlegung der Konstante *factor*. Der Umrechnungsfaktor ist 0,6214. Bei Java ist allerdings das Dezimaltrennzeichen der Punkt.

Listing 3.4 Festlegung des Umrechnungsfaktors

```
public class MainActivity extends Activity {
    private final static double factor = 0.6214;
    ...
```

2. Als Nächstes wollen wir auf das Betätigen des Buttons reagieren. Da Sie schon bei der Erstellung des Buttons (*berechnen*) das *OnClick-Ereignis* mit *onClick* definiert haben, müssen wir nur noch die Methode hinzufügen, die *onClick* von Views auswertet. Jedoch genügt es nicht, nur eine Methode hinzuzufügen. Es würde nichts passieren. Damit die *Activity* auf *OnClick-Ereignisse* reagiert, müssen wir das Interface *View.OnClickListener* implementieren. Vereinfacht ausgedrückt, kann man sagen, das Interface macht das *onClick-Ereignis* der View, dem Button und der Klasse *MainActivity* bekannt, und innerhalb der Klasse *MainActivity* wird darauf reagiert. Zunächst implementieren Sie das Interface *View.OnClickListener* einfach durch *implements OnClickListener*. Danach folgt in Listing 3.6 die Methode für die Behandlung des *onClick-Ereignisses*. Die Annotation *@Override* kennzeichnet, dass eine Methode des Interfaces *View.OnClickListener* überschrieben wird. Als Parameter erhält diese Methode die View, die dieses Ereignis auslöst. Die mit *if* beginnende Zeile überprüft, ob dies auch der richtige Button ist. Weiter wird jetzt noch nichts ausgeführt.

Listing 3.5 OnClickListerner implementieren

```
public class MainActivity extends Activity implements OnClickListener {
    ...
}
```

Listing 3.6 Methode für das onClick-Ereignis des Buttons „berechnen“

```
@Override
public void onClick(View v) {
    if(v.getId() == R.id.button_berechnen){
        }
}
```

3. Nun wollen wir die Methode zur Umrechnung von Kilometern in Meilen erstellen. Die Methode zur Berechnung der Meilen führt Folgendes aus:

- Bekanntmachen des Eingabe-Steuerelements
- Einlesen der Eingabe in die Variable `eingabeKm`
- Berechnung der Meilen und Zuweisung des berechneten Wertes der Variable `miles`
- Bekanntmachen des Ausgabe-Steuerelements
- Format festlegen
- Berechneten Wert anzeigen (ausgeben)

Listing 3.7 Methode zur Berechnung der Meilen

```
private void calculateMiles() {
    /* Eingabe */
    EditText eingabe = (EditText) findViewById(R.id.editText_eingabe);
    final double eingabeKm = Double.parseDouble(eingabe.getText().toString());
    /* Berechnung */
    double miles = eingabeKm * factor;
    /* Ausgabe */
    TextView ausgabe = (TextView) findViewById(R.id.textView_ausgabe_miles);
    DecimalFormat df = new DecimalFormat("0.000");
    ausgabe.setText(df.format(miles));
}
```

4. Es folgt die Methode `calculateMiles()` ausführen, bei Button `onClick`. Damit die Methode `calculateMiles()` aus Listing 3.7 bei einem `onClick` Ereignis des Buttons ausgeführt wird, muss die Methode in der Abarbeitung des `onClick-Ereignisses` der View aufgerufen werden. Dies erfolgt durch Hinzufügen des Aufrufs der Methode `calculateMiles();` in der Methode `onClick(View v)`.

Listing 3.8 Methode `onClick(View v)` mit Umrechnung von Kilometern in Meilen

```
@Override
public void onClick(View v) {
    if (v.getId() == R.id.button_berechnen) {
        calculateMiles();
    }
}
```

5. Als Letztes soll der Menüeintrag *berechnen* Funktionalität erhalten. Bei Menüs, in diesem Fall der *ActionBar*, wird anders vorgegangen. Der Projektassistent erstellte die notwendige Methode zur Erzeugung der *ActionBar*. Sie müssen nun nur noch die Methode `onOptionsItemSelected(MenuItem item)` aus Listing 3.9 hinzufügen, die der *ActionBar* Funktionalität verleiht. Diese Methode wird immer dann ausgeführt, wenn in der *ActionBar* ein Menüpunkt ausgewählt wird.

Listing 3.9 Methode Menü: `onOptionsItemSelected(MenuItem item)`

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.action_berechnen:  
            calculateMiles();  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

3.1.5 Die Struktur von Android-Projekten

Der Android-Projektassistent hat die komplette Struktur des Projekts angelegt. Wir wollen nun näher darauf eingehen. Denken Sie daran, dass die Ansicht im Package Explorer eine logische Ansicht ist. Möchten Sie die Dateistruktur auf der Festplatte sehen, müssen Sie die *Navigator View* in Eclipse öffnen.

Das Projekt enthält im Package Explorer jetzt folgende Nodes (Knoten):

- **src (source):** Dieser Node enthält alle Sourcecode-Dateien der Anwendung. Der Sourcecode von Android wird wie in Java in Packages (Paketen) organisiert.
- **gen (generated):** Der Inhalt dieses Nodes enthält ebenfalls Java-Quellcodedateien, jedoch werden diese Dateien automatisch vom ADT generiert. Diese Dateien werden vom ADT verwaltet und sollten deshalb nicht verändert werden. Betrachten Sie einmal die Klasse *R.java*. Sie werden feststellen, dass in dieser Klasse alle inneren Klassen und Dateien (Klassen innerhalb einer Klasse) den gleichen Namen haben wie die Ordner oder Dateien unterhalb von */res*.



PRAXISTIPP: Manchmal kann es vorkommen, dass die Datei *R.java* nicht fehlerfrei neu erstellt wird, und Sie viele Fehler angezeigt bekommen.

Oft lösen Sie das Problem, wenn Sie in Eclipse den *Autobuild* für Projekte abschalten, das Package in *gen* löschen und das Projekt neu erstellen.

- **Android 4.2.2:** Dies ist die verwendete Android-Version. Hier finden Sie eine kompilierte Java-Datei, die alle Klassen der Android-Version und sogar noch einige mehr enthält. Dies ist der komplett verfügbare Sprachumfang von Android und Java, der Ihnen für die Programmierung zur Verfügung steht.
- **Android Dependencies:** Dieser Node enthält Verweise auf Abhängigkeiten Ihrer Anwendung. Die darin enthaltene *android-support-v4.jar* ist ebenfalls eine kompilierte Java-Datei,

die Support-Klassen für Versionen vor Android 4.0 bereitstellt. Damit wurde die Möglichkeit geschaffen, bestimmte mit Android 4 eingeführte Funktionalitäten, früheren Versionen von Android zur Verfügung zu stellen, wie z. B. Fragments. Diese Bibliothek befindet sich in Wirklichkeit im Ordner *libs*, und hier wird nur darauf verwiesen.

- **assets:** In diesem Node werden alle Dateien abgelegt, die nicht mit den Android-Standardmechanismen geladen werden können. Dies könnten z. B. Klingelföne, Hintergrundbilder oder Schriftarten sein. Diese Dateien werden meist nicht von der Anwendung direkt angesprochen, sondern sollen bei der Installation der App ins System installiert werden.
- **bin (binary):** Der Node *binary* enthält den Output des Compilers und des ADT-Resourcen-Tools. Sie finden neben der *Apk*-Datei (Android Package) auch das Manifest und Ressourcen sowie weitere Dateien.
- **libs (libraries):** In diesem Ordner werden, wenn nötig, zusätzliche Bibliotheken abgelegt. Es genügt allerdings nicht einfach nur eine Bibliothek hier abzulegen.
- **res (resources):** Der Node *res* nimmt alle Ressourcen auf, die sich schwer oder gar nicht im Code darstellen lassen. Vereinfacht kann man sagen, alle Dateien, die kein Java-Quelltext sind, wie Bilder, XML-Definitionsdateien für Styles, Layouts, Themes oder Textdefinitionen, werden hier abgelegt. Android verfügt über ein mächtiges Ressourcen-Management-System, das abhängig von der Hardware-Konfiguration des Gerätes diese Ressourcen lädt. Die Organisation von Ressourcen unterhalb von *res* erfolgt nach Art der Ressource in Unterordnern. Die Grundordner unterhalb von *res* sind: *drawable*, *layout*, *menu* und *values*. Diese Ordnernamen können mit Kennzeichnern für unterschiedliche Android Versionen oder Hardwareeigenschaften, wie z. B. *drawable-hdpi*, erweitert werden. *Drawables* sind in Android nicht nur Bilder, *drawables* können auch per XML definiert werden. Der Ordner *drawable-hdpi* enthält Ressourcen, *drawables* (Bilder, Icons, XML-Dateien), für die *hdpi*-Auflösung. Im Ordner *layout* finden Sie alle XML-Layouts der Oberfläche der App. In *menu* werden die Menüs der Anwendung per XML deklariert. Im Ordner *values* werden alle Wertedefinitionen in *xml*-Dateien abgelegt. Diese Wertdefinitionen können z. B. *Strings* (*strings.xml*), *Farben* (*colors.xml*), *Attribute* (*attr.xml*) oder auch ganze Wertgruppen in *Arrays* (*array.xml*) sein. Weitere Ressourcen-Ordner sind *anim*, *animator*, *raw* und *xml*. In den *raw*-Ordner werden jegliche Arten von Rohdaten abgelegt. Dies können Sounds, Bilder etc. sein, die von der Anwendung benötigt werden. Der Ordner *xml* kann generische XML-Daten enthalten.



HINWEIS: Im Ordner */res* dürfen die Dateinamen nur aus Kleinbuchstaben, dem Unterstrich und Ziffern bestehen. Eine andere Namensgebung ist in diesem Ordner nicht erlaubt.

Weiterhin enthält unser Projekt noch folgende Dateien:

- **AndroidManifest.xml:** Die Android Manifest Datei ist die zentrale Beschreibung der Anwendung, hier werden unter anderem Einstellungen zur Version, Programmbestandteile, Anwendungsrechte und noch weiteres mehr festgelegt. Wir widmen daher dem Manifest einen eigenen Abschnitt.
- **ic_launcher-web.png:** Dies ist das Anwendungs-Icon für den Google Play Store. Jede Anwendung benötigt mindestens ein hochauflösendes Bild zur Anzeige im Play Store.

Dieses Bild ist 512×512 px groß, ausführliche Informationen zu Launcher-Icons finden Sie unter http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html#size.

- **proguard-project.txt:** Diese Datei hat keinerlei Bedeutung, sie enthält nur Informationen für eine Konfigurationsdatei des ProGuard Tools. Weiterführende Informationen zu ProGuard finden Sie unter: <http://developer.android.com/tools/help/proguard.html>.
- **project.properties:** In dieser Datei werden die Zielversion des API-Level oder auch zusätzliche Bibliotheken referenziert.

Das Android-Manifest, **AndroidManifest.xml**

Ein Manifest ist laut Wikipedia eine öffentliche Erklärung von Zielen und Absichten, und dies trifft auch auf das Android-Manifest zu. Das Android-Manifest beschreibt die App gegenüber dem Betriebssystem, aus welchen Komponenten sie besteht, wie sie heißt, die Version der App oder welche Rechte die App benötigt, um fehlerfrei zu funktionieren. Das Android-Manifest ist eine *xml*-Datei, die bei der Kompilierung in binäres XML übersetzt wird.

Wenn Sie die *AndroidManifest.xml*-Datei öffnen, wird Ihnen zuerst die Ansicht in der grafischen Oberfläche präsentiert. Der Editor präsentiert Ihnen im unteren Bereich fünf Karteireiter. Am Anfang ist es einfacher, die Einstellungen über den grafischen Editor vorzunehmen, im Laufe der Zeit als Android-Entwickler werden Sie jedoch viele Einstellungen direkt im XML-Code vornehmen.

- **Manifest:** Der Package-Name ist derselbe wie Sie ihn bei der Projektanlage angegeben haben. Zur Erinnerung, der Package-Name identifiziert Ihre Anwendung eindeutig im Google Play Store. Der *Versions code* wird ganzzahlig (*int*) angegeben, und bei einem Update muss dieser erhöht werden. Der *Version name* ist die Versionsbezeichnung und sollte bei einem Update ebenfalls erhöht werden. Der *Version name* wird im Google Play Store und auch in der App-Übersicht auf dem Gerät angezeigt. *Shared user id* und *Shared user label* werden benötigt, wenn Sie mehrere Anwendungen erstellt haben und diese untereinander kommunizieren sollen. Diese werden dann auch in einer gemeinsamen *Sandbox* auf dem Gerät ausgeführt. Bei *Install location* sollten Sie genau überlegen, was Sie hier einstellen. Hier wird entschieden, wo die App installiert wird. Die Einstellung *auto* überlässt Android die Entscheidung, wo die App installiert wird, *internalOnly* ist der interne Gerätespeicher, und *preferExternal* installiert die App auf der SD-Card, sofern verfügbar. Sie sollten grundsätzlich dem User die Freiheit lassen, wo er seine App installiert, allerdings gibt es Gründe, die Sie zwingen, den internen Speicher vorzuschreiben. Sollte Ihre App z.B. Widgets oder Hintergrunddienste verwenden, und während des Betriebes wird die SD-Karte entfernt, so stürzt die App ab, ohne erkennbaren Grund für den User. Im Bereich *Manifest Extras* finden Sie den Eintrag *Uses Sdk*. Dieser zeigt Ihnen wieder die Optionen *Min SDK*, *Target SDK* und *Max SDK*, die Sie bei der Projekterstellung gewählt haben. Durch *Min SDK* werden alle Geräte im Google Play Store herausgefiltert, deren API-Level kleiner *Min SDK* ist. Die *Target SDK version* ist wichtig für die Ausführung auf den Geräten. Ist der API-Level auf dem Gerät höher, so wird die App in einem Kompatibilitätsmodus ausgeführt, da Android sehr gut abwärtskompatibel ist. *Max SDK version* werden Sie wohl selten oder gar nicht einsetzen. Diese Einstellung wirkt wie *Min SDK version*, alle Geräte die einen höheren API-Level haben, können diese App nicht installieren und

bekommen diese im Play Store nicht angezeigt. In *Manifest Extras* werden die Anforderungen ihrer App an das Gerät definiert, neben den schon aufgeführten.

- **Application:** Dieser Karteireiter beschreibt alle Mechanismen, aus denen die Anwendung besteht. Im Bereich *Application Attributes* können Sie die Oberfläche und die Ausführungsumgebung der Anwendung einstellen. Labels, Icons und Logos können Sie Ressourcen aus dem Ressourcen-Ordner zuweisen. Der Bereich *Application Nodes* ist für die Ausführung der App wichtig. Hier müssen Sie alle Komponenten, die in der Anwendung genutzt werden, angeben. Zu den Komponenten zählen unter anderem *Activities*, *Broadcast Receiver*, *Services* oder *Content Provider*. Komponenten, die hier nicht eingetragen sind, können in der Anwendung nicht genutzt werden und Ihre App stürzt einfach ab. Im Debugger erhalten Sie dann lediglich die Meldung *Komponente oder Resource wurde nicht gefunden*. Jede Activity kann abweichend von der allgemeinen Konfiguration für sich konfiguriert werden. Diese Einstellungen finden Sie neben *Application Nodes*, sobald Sie eine *Activity* markieren.
- **Permissions:** Damit eine Anwendung auf Ressourcen außerhalb ihrer Sandbox oder Systemdienste zugreifen kann, müssen ihr diese *Permissions (Berechtigungen)* explizit erlaubt werden. Diese Berechtigungen, die die App für die fehlerfreie Ausführung benötigt, z.B. Dateien zu speichern, auf Kontakte oder auf das Internet zuzugreifen, werden in diesem Bereich eingestellt. Die *Permissions* werden auch im Google Play Store angezeigt und informieren den User darüber, auf welche Daten, Dienste oder Hardware die Anwendung zugreift. Geben Sie Ihrer Anwendung nur die Rechte, die benötigt werden, sonst erzeugen Sie unnötige Sicherheitslücken. Weiterhin sollten Sie mit diesen Einstellungen sehr transparent gegenüber dem Nutzer der Anwendung umgehen, um Vertrauen zu schaffen.
- **Instrumentation:** Der Bereich Instrumentation verwaltet Klassen oder Komponenten, mit denen Sie die Anwendung automatisch testen wollen, Profilings durchführen oder Benchmarks unterziehen.
- **AndroidManifest.xml:** Dies ist die XML-Ansicht des Manifestes, hier können Sie manuelle Einstellungen vornehmen. Verändern Sie die XML-Struktur der Datei nicht, der Google Play Store ist sehr empfindlich, und es kann passieren, dass Ihre App abgewiesen wird.

■ 3.2 Die App im Emulator und auf einem Gerät testen

In diesem Abschnitt erläutern wir die Einrichtung eines virtuellen Gerätes und zeigen auf, wie Sie die Konfigurationen von Hardware-Herstellern nutzen können. Emulatoren verwalten Sie mit dem *AVD-Manager*, der vom *Sdk* mitgeliefert wird. Den AVD-Manager können Sie direkt aus Eclipse heraus per Klick auf **WINDOW/ANDROID VIRTUAL DEVICE MANAGER** starten oder durch Doppelklick auf die **AVD MANAGER.EXE** im *Sdk*-Ordner.

3.2.1 Emulator definieren

Die Erstellung eines virtuellen Gerätes ist recht einfach und sollte keine Probleme verursachen. Bevor Sie jetzt ein neues virtuelles Device (Gerät) anlegen, wechseln Sie kurz auf den Tab *Device Definitions*. Sie finden hier eine Auflistung von Gerätedefinitionen, die als Muster für die Erstellung eines virtuellen Gerätes dienen. Es sind sowohl Vorlagen, die auf den Nexus-Geräten von Google beruhen, wie auch generische. Diese Vorlagen können Sie kopieren, nach Ihren Vorstellungen anpassen oder eigene erstellen. Sie sollten die Originalvorlagen nicht anpassen, da diese bei einem Update des SDK unter Umständen überschrieben werden. Kopieren Sie stattdessen eine Vorlage und vergeben einen eigenen Namen.

Um ein neues virtuelles Gerät anzulegen, wechseln Sie wieder auf den Tab *Android Virtual Devices* und starten die Erstellung durch **New...** an der rechten Seite.

1. Vergeben Sie einen Namen für den Emulator. Es sind keine Sonderzeichen und Leerzeichen erlaubt. Wählen Sie einen Namen, der auch wichtige Infos, wie die API oder das Gerät, enthält.
2. Nun wählen Sie das Gerät aus. Google liefert ja eine Menge vorgefertigter Geräte-Vorlagen mit. Sie können aus Emulatoren von realen Geräten oder nach Display-Größe, Orientierung und Auflösung auswählen.
3. Jetzt wählen Sie die *Target Version* aus. Diese sollte der *Target Version* der Anwendung entsprechen, außerdem sollten Sie die generischen Definitionen bevorzugen, wenn die Anwendung für alle möglichen Geräte getestet werden soll. Der Unterschied der Target Api Google und Android ist, dass die Google-API verschiedene Google-Dienste, wie z.B. Maps, Local oder Navigation, mitbringt.

Sollen Eingaben über Ihre Computertastatur möglich sein, muss *Keyboard* aktiviert sein.

Bei älteren Geräten waren Hardware-Tasten vorhanden. Wenn Sie *Skin* deaktivieren, werden keine externen Hardware-Tasten mehr angezeigt und Ihr Emulator wirkt wie ein aktuelles Gerät.

Front Camera/Back Camera: Diese Einstellungen sind selbsterklärend.

Memory Options: Mit der Option *RAM* legen fest, wie viel Arbeitsspeicher das virtuelle Device besitzt. Der hier eingestellte Wert wird auch in Ihrem System reserviert. Mit *VM Heap* bestimmen Sie die Größe des reservierten Speichers für die *Dalvik Virtual Machine* und Android-Systemprozesse. Sie sollten *RAM* nicht größer als 768 MB wählen, und für *VM Heap* ist 32 MB ein guter Durchschnittswert. Einstellungen von 1.024 MB RAM und VM Heap 64 führten auf einigen Testsystemen immer wieder zu Problemen, mehr ist nicht immer besser.

Internal Storage: Diese Einstellung ist analog der Speicherkapazität von Geräten zu sehen. Sie legen hier fest, wie viel Speicherplatz auf dem virtuellen Device zur Verfügung stehen soll. Sie sollten beachten, dass die hier eingestellte Größe auch auf Ihrer Festplatte reserviert wird. Bei sehr vielen AVD kann der Festplattenplatz schnell knapp werden.

SD Card: Diese Einstellung ist gleich der von *Internal Storage*, gilt allerdings für die Emulation der SD-Speicherkarte, die die meisten Android-Geräte verwenden können.

Emulation Options: Diese zwei Optionen beschleunigen den Start oder die Ausführung des virtuellen Gerätes. Bei der Option *Snapshot* wird ein solcher beim Beenden des virtuellen

Gerätes angelegt, und beim nächsten Start kann direkt vom letzten Abbild gestartet werden. Der Start der AVD ist dann schneller. Mit der Option *Use Host GPU* aktivieren Sie die Beschleunigung durch zusätzliche Nutzung des Prozessors Ihrer Grafikkarte. Sie können immer nur eine Option nutzen, welche auf Ihrem System die bessere ist, müssen Sie ausprobieren.

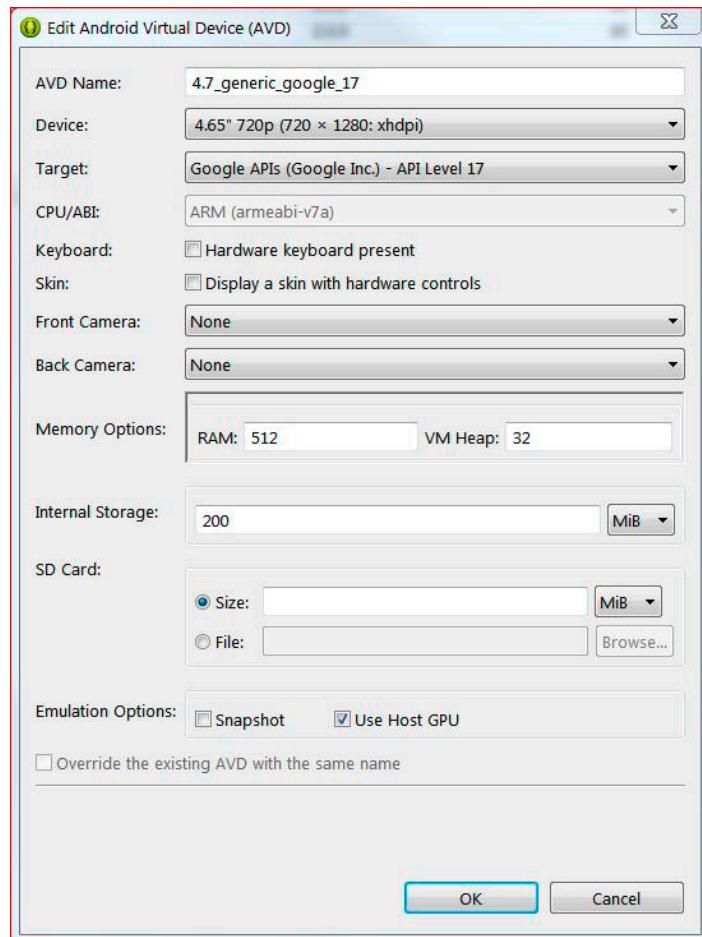


Bild 3.16 Vollständig definiertes virtuelles Gerät

Viele Hersteller von Android-Geräten bieten auch Erweiterungen an, um die Entwicklung von Anwendungen auf diesen Geräten speziell zu testen oder um Anwendungen zu entwickeln, die die Herstelleranpassungen nutzen. Dafür stellen diese Hersteller so genannte Add-ons für den Emulator und das SDK bereit.



HINWEIS: Den Pfad zu den Konfigurationen und Abbildern der erstellten AVD sehen Sie im AVD-Manager, Tab Android Virtual Devices in der ersten Zeile des Fensters.

3.2.2 Tastaturbefehle für den Android-Emulator

Sie können den Emulator auch über die Tastatur steuern. Im anschließenden Kasten haben wir die Tastaturbefehle aufgeführt.



3.2.3 Die App auf einem Emulator oder Gerät starten

Ihre erste App ist fertig. Sie haben eine AVD definiert, nun kann diese im Emulator oder Ihrem Gerät getestet werden. Die Installation einer App im Emulator oder einem Gerät ist ähnlich, daher beschreiben wir in diesem Abschnitt nur die Vorgehensweise, um die fertige Testanwendung auf dem Emulator auszuführen. Gehen Sie dazu wie folgt vor:

1. Klicken Sie oben in der Symbolleiste von Eclipse auf RUN, wie in Bild 3.17 abgebildet, und wählen im nächsten Fenster DEBUG. Wie so oft in Eclipse, können Sie diese Aktion mit der Taste F11 oder durch Klicken auf den Käfer in der Symbolleiste ausführen.

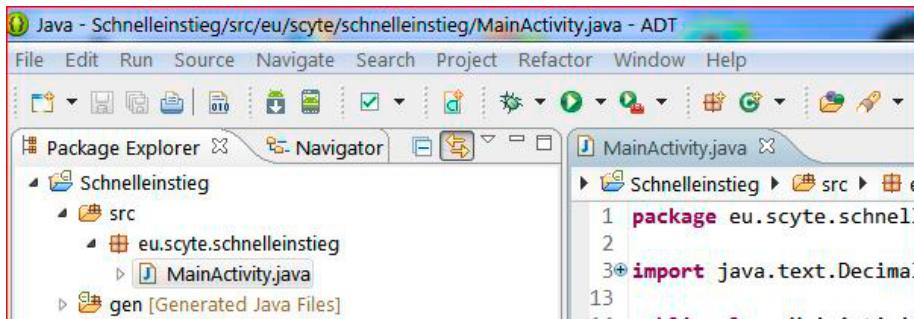


Bild 3.17 App im Emulator starten

2. In dem jetzt erscheinenden Fenster (Bild 3.18) wählen Sie die Option *Android Application* und klicken auf OK.

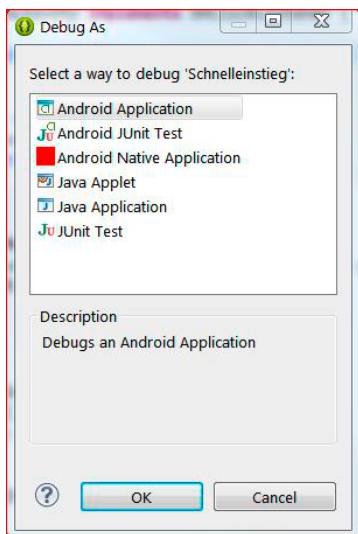


Bild 3.18
Fenster Debug As

3. Sollten Sie noch keinen Emulator gestartet haben, startet der AVD-Manager diesen automatisch. Das ADT wählt automatisch eine AVD aus, die der SDK-Version entspricht. Der Start des Emulators kann eine Weile dauern.
4. Nachdem der Emulator gestartet ist, sehen Sie den klassischen Android-Startbildschirm. Entsperren Sie Ihr virtuelles Gerät. Da die App im Hintergrund installiert und gestartet wurde, sehen Sie nach dem Entsperren die Anwendung und können sofort testen.
5. Klicken Sie nun in das *Eingabe-Steuerelement* und geben einen Wert ein. Danach klicken Sie auf den Button **BERECHNEN**. Sie erhalten die Anzeigen der berechneten Meilen. Testen Sie ebenfalls den Menüeintrag **BERECHNEN** oben in der *ActionBar*, Sie sollten allerdings einen neuen Wert eingeben.
6. Sollte der Emulator zu groß dargestellt werden, müssen Sie den Emulator noch einmal durch Schließen des Emulator-Fensters beenden. Öffnen Sie den AVD-Manager in Eclipse, WINDOW/ANDROID VIRTUAL DEVICE MANAGER, markieren Ihre AVD und klicken auf der rechten Seite auf START. Aktivieren Sie die Option *Scale display...*, ändern Sie die Werte in *Screen Size (in)* und *Monitor dpi*. Das Fragezeichen neben *Monitor dpi* unterstützt Sie bei der Ermittlung der *dpi* Ihres Monitors (Bild 3.19). Sollten Sie eine Fehlermeldung erhalten, wie „*Unable to load VM from snapshot*“, müssen Sie die Option *Launch from Snapshot* deaktivieren.

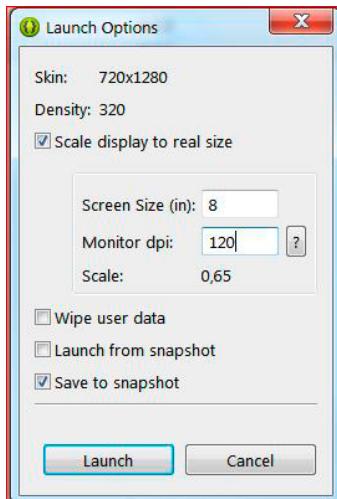


Bild 3.19
Emulator-Start-Optionen

- Jetzt sollten Sie sich einmal die vom ADT angelegte Debug-Konfiguration der Anwendung ansehen. Um die Debug-Konfigurationen zu öffnen, klicken Sie wieder auf RUN/DEBUG CONFIGURATIONS... oder auf den Pfeil nach unten, neben dem Käfer in der Symbolleiste. Wie in Bild 3.20 zu sehen, wurde vom ADT automatisch eine Konfiguration zum Debuggen für das Projekt angelegt. Auf dem Tab *Android* sehen Sie das ausgewählte Projekt und welche Activity beim Start der Anwendung auf die AVD geladen werden soll. Es wird hier die Start-Activity der Anwendung gestartet. Sie können hier aber auch gezielt eine andere Activity einstellen. Dies ist sehr praktisch, da die Navigation zur Activity entfällt und man sich auf den Bereich konzentrieren kann, der gerade bearbeitet wird.

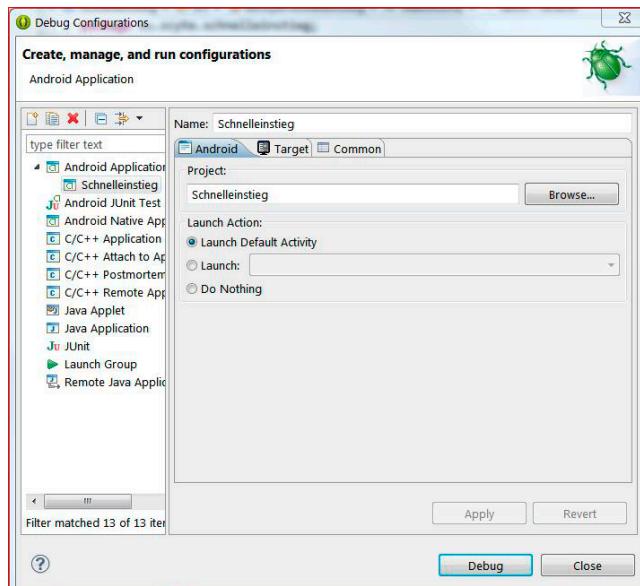


Bild 3.20 Debug-Konfiguration: Schnelleinstieg Tab Android

8. In Bild 3.21 sehen Sie den Tab *Target*. Die Einstellung *Always prompt...* bedeutet, dass Sie jedes Mal beim Starten des Debuggens gefragt werden, welches Device genutzt werden soll. Sie können dann in einem separaten Fenster aus gerade aktiven Geräten, Emulatoren und auch aus angeschlossener Hardware wählen, oder wenn nichts verfügbar ist, eine AVD starten. Aktivieren Sie die Einstellung *Automatic...*, so wählt das ADT automatisch den am besten geeigneten Emulator. Sie können jedoch auch eine AVD in der Übersicht auswählen, die genutzt werden soll. Haben Sie mehrere aktive Geräte zur Verfügung, erhalten Sie auch das Geräteauswahlfenster. Wir bevorzugen die Option *Always prompt...* Sie können auch direkt aus diesem Fenster die Debug-Sitzung starten, klicken Sie hierzu auf **DEBUG** und wählen dann Ihr Gerät.

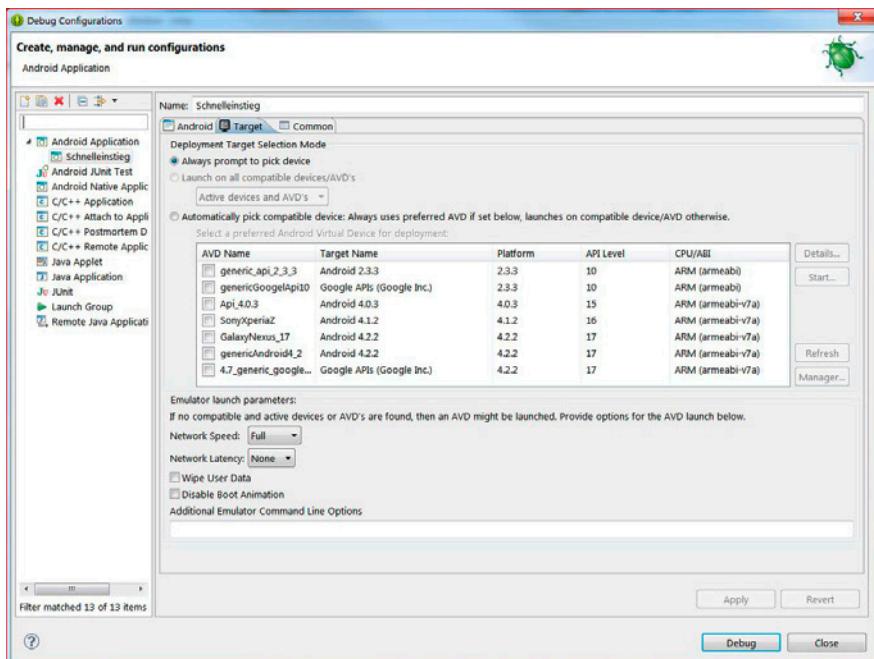


Bild 3.21 Debug-Konfiguration: Schnelleinstieg Tab Target

9. Schließen Sie jetzt Ihr Android-Gerät an, um die Anwendung auf Ihrer Hardware zu testen. Starten Sie nun eine neue Debug-Sitzung und wählen Ihr Gerät aus. Im Hintergrund wird die Anwendung auf Ihrem Gerät installiert und gestartet.

■ 3.3 Crash-Kurs in Java

Java ist die primäre Programmiersprache für die Entwicklung von Anwendungen beziehungsweise Apps für Android. Java orientiert sich in vielen Teilen an C und C++ und erbt vom Smalltalk, besitzt aber auch exklusive Sprachmerkmale, wie z.B. den eigenen Datentyp **boolean**.

Dieser Abschnitt versteht sich als Schnelleinführung in die wichtigsten Merkmale der Programmiersprache Java. Er richtet sich an Entwickler, die bereits mit einer anderen objekt-orientierten Sprache grundlegend vertraut sind. So werden allgemeine Eigenschaften von Programmiersprachen wie Variablen, Methoden oder Klassen nicht grundlegend erläutert, sondern es werden lediglich deren Syntax und Verwendung in Java zusammengefasst.



HINWEIS: In diesem Abschnitt werden nur die wichtigsten Sprachmerkmale von Java vorgestellt, die für das Verständnis der Code-Beispiele in den nachfolgenden Kapiteln dieses Buches Voraussetzung sind.

Java verfügt über zahlreiche weitere Sprachmerkmale. Hierzu gehören beispielsweise:

- Objektorientierung
- Kapselung
- Case Sensitive
- Vererbung, aber keine Mehrfachvererbung
- Delegates
- Enumerationen
- Interfaces
- Reflection
- Annotationen
- Collections
- Autoboxing
- Definition eigenständiger Datentypen

Einige dieser Features werden im weiteren Verlauf dieses Buches noch verwendet und dann auch kurz erläutert.

Einige grundlegende Hauptmerkmale der Syntax von Java sind:

- Anweisungen können über mehrere Zeilen gehen und werden mit einem Semikolon abgeschlossen.
- Methodenaufrufe ohne Parameter innerhalb einer Klasse erfolgen durch den Namen, gefolgt von runden Klammern.
- Parameter von Methoden werden in runden Klammern mit Typen und Namen angegeben, und mehrere Parameter werden durch Kommas voneinander getrennt.
- Objektverweise beginnen mit einem Stern-Symbol.
- Zeichenketten (Strings) werden in doppelte Anführungszeichen gesetzt.

3.3.1 Werte einer Variablen zuordnen

Die Anweisung zur Definition von Variablen in Java setzt sich aus dem Typ der Variablen sowie ihrem Namen zusammen. Die Zuweisung eines Wertes erfolgt mit dem Gleichheitszeichen als Operator:

```
int zahl;
```

```
zahl = 5;
```

oder

```
int zahl = 5;
```

Da Java an C anlehnt, können, wie in dem vorausgegangen Code-Beispiel zu sehen, die Standardtypen aus C, wie *int*, *double*, *long* und viele weitere, genutzt werden. Darüber hinaus verfügt Java auch über eigene Datentypen sowie über viele weitere Klassen.

Primitive Datentypen in Java:

- **ganze Zahlen:** byte, short, int, long
- **Gleitkommazahlen:** float, double
- **Textzeichen:** char
- **Wahrheitswerte:** boolean, ist in Java ein eigener Datentyp und kann nur *true* oder *false* sein.

Die Datentypen **int** und **double** sind in Java die Standarddatentypen für Ganz- bzw. Gleitkommazahlen.



HINWEIS: Variablen in **Methoden** müssen immer initialisiert werden, das heißt, Variablen innerhalb einer Methode müssen immer einen Startwert haben, z.B. `int zahl = 0;`

3.3.2 Bezeichner und Schlüsselwörter in Java

Bei der Erstellung von Java-Programmen müssen Sie für alles Namen vergeben, so genannte Bezeichner. Bezeichner sind nichts anderes als die Namen für Klassen, Methoden, Attribute, Variablen und Konstanten. Grundsätzlich sind Sie frei in der Wahl der Bezeichner, allerdings müssen Sie einige Regeln und Konventionen beachten, damit Ihre Anwendung überhaupt kompiliert wird und andere Ihren Code lesen und verstehen können. Auch ist es einfacher, den eigenen Code nach einer Weile zu verstehen, wenn Sie sich an die Code-Konventionen halten.

Sie sollten daher folgende Regeln beachten:

- **Bezeichner** dürfen nicht mit einer Zahl beginnen und sollten sprechende Namen haben. Außerdem dürfen Bezeichner keine Sonderzeichen enthalten, `getName` sagt mehr als `getN`.

- **Attribute und Namen** beginnen mit einem kleinen Buchstaben, und bei zusammengesetzten jedes weitere Wort mit einem Großbuchstaben, zum Beispiel noteType (nT wäre schlecht lesbar).
- **Konstanten** sollten aus Großbuchstaben bestehen und zusammengesetzte Worte werden mit einem Unterstrich getrennt, z. B. MAX_LENGTH.
- **Methoden** werden wie Attribute oder Namen geschrieben: `getLength()`. Jedes beliebige Zeichen ist erlaubt, darf aber nicht mit einer Zahl beginnen. Sonderzeichen wie „.,[, } sind nicht erlaubt.
- **Klassen** sollten immer mit Großbuchstaben begonnen werden, z. B. Motorrad, Note, Reminder, BroadcastReceiver.
- Java ist **case-sensitive**, das bedeutet, dass immer zwischen Groß- und Kleinschreibung unterschieden wird, noteType ist nicht das Gleiche wie notetype. Dies sind unterschiedliche Bezeichner!
- Java-Schlüsselwörter sind grundsätzlich **nicht** als Bezeichner erlaubt.

Tabelle 3.1 Java-Schlüsselwörter

<code>abstract</code>	<code>continue</code>	<code>for</code>	<code>new</code>	<code>switch</code>
<code>assert</code>	<code>default</code>	<code>goto</code>	<code>package</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>private</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>protected</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>public</code>	<code>throws</code>
<code>case</code>	<code>enum</code>	<code>instanceof</code>	<code>return</code>	<code>transient</code>
<code>catch</code>	<code>extends</code>	<code>int</code>	<code>short</code>	<code>try</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>static</code>	<code>void</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>strictfp</code>	<code>volatile</code>
<code>const</code>	<code>float</code>	<code>native</code>	<code>super</code>	<code>while</code>



HINWEIS: Folgende Literale sind nicht erlaubt: true, false und null.

Auch Google hat mittlerweile einige Konventionen herausgegeben, wie der Code gestaltet werden soll (siehe <http://developer.android.com/training/articles/perf-tips.html>).

3.3.3 Klassen und Objekte in Java

Ein wesentliches Merkmal einer objektorientierten Programmiersprache sind die Klassen. Klassen kann man als eine Art Bauplan von Objekten verstehen. In Klassen werden Eigenschaften und Fähigkeiten von Objekten definiert. Die Basisklasse aller Klassen ist die Klasse **Object**.

Die Eigenschaften einer Klasse bezeichnet man als *Attribute*, die Fähigkeiten als *Methoden* einer Klasse. Außerdem enthalten Java-Klassen fast immer so genannte **Konstruktoren**, die bei der Initialisierung eines Objekts aufgerufen werden. Ein **Konstruktor** ist eine spezielle Methode, die den Namen der Klasse tragen und beim Erzeugen von Objekten der Klasse über das Schlüsselwort **new** aufgerufen werden. Konstruktoren von Klassen haben immer den Rückgabetyp **void**. Der in Listing 3.10 deklarierte Konstruktor `public Motorrad()` ist ein Default-Konstruktor. Wird dieser in der Klassendefinition nicht angegeben, so erzeugt Java ihn automatisch. Konstruktoren werden eingesetzt, um bei der Initialisierung dafür zu sorgen, dass bestimmte notwendige Eigenschaften von Objekten festgelegt werden müssen.

Listing 3.10 Eine Beispielklasse: Motorrad mit Attribut, Konstruktor und Methode

```
public class Motorrad{
    //Attribute
    private int geschwindigkeit;
    private String typ;
    //Konstruktor
    public Motorrad(){
    }
    //Methoden
    public void beschleunigen(){
        geschwindigkeit = geschwindigkeit + 1;
    }

    public void bremsen(){
        geschwindigkeit = geschwindigkeit - 1;
    }
}
```

Diese Klasse beschreibt, wie Objekte des Typs Motorrad auszusehen haben und was das Motorrad kann, und enthält außerdem den Default-Konstruktor. In Listing 3.11 ist ein Beispiel für die Klasse Motorrad mit einem Konstruktor aufgeführt, der die Zuweisung des Attributs *typ* bei der Erzeugung eines neuen Objektes erzwingt.

Listing 3.11 Klasse Motorrad mit parametrisiertem Konstruktor

```
public class Motorrad{
    //Attribute
    private int geschwindigkeit;
    private String typ;
    //Konstruktor
    public Motorrad(String typ){
        this.typ = typ;
    }
}
```

3.3.4 Objekte erzeugen und initialisieren

Bevor Sie auf ein Objekt zugreifen können, müssen Sie es zuerst initialisieren.

Ein Objekt wird folgendermaßen erzeugt:

```
Klasse meinObject = new Klasse();
Motorrad meinMotorrad = new Motorrad();
```

Hier wird eine *Objektreferenz* *meinMotorrad* der *Klasse* Motorrad erzeugt und initialisiert. Enthält die Klasse Motorrad einen Konstruktor wie in Listing 3.11, muss die Initialisierung eines neuen Objektes mit dem Argument für den Parameter *typ* erfolgen.

Beispiel:

```
Motorrad meinMotorrad = new Motorrad("Touring");
```

3.3.5 Methoden von Objekten aufrufen

Der Aufruf einer Objektmethode ohne Parameter erfolgt durch den Aufruf der Objektreferenz, gefolgt von einem Punkt und dem Namen der Methode. Bevor Sie jedoch eine Objektmethode aufrufen können, müssen Sie ein Objekt initialisieren:

```
//Objekt initialisieren
Klasse meinObject = new Klasse();
//Objektmethode aufrufen
meinObject.Methode();
```

Methoden können natürlich auch Werte entgegennehmen. Ob eine Methode einen oder mehrere Werte entgegennehmen kann, wird bei der Methodendeklaration festgelegt. Eine Methode, die Werte entgegen nimmt, wird als *parametrisierte* Methode bezeichnet. Ein Parameter wird immer durch Typ und Bezeichner festgelegt, mehrere Parameter werden untereinander durch Kommas getrennt deklariert, die Kurzschreibweise `int a, b;` ist nicht erlaubt.

```
void meineParametrisierteMethode(int zahl, String text){
}
```

Ruft man eine solche Methode auf, so bezeichnet man die übergebenen Werte als Argumente. Beim Aufruf einer solchen parametrisierten Methode sind immer alle Argumente (Parameter) anzugeben, die Argumente müssen vom Typ des Parameters sein.

```
meineParametrisierteMethode( 10, "Hallo") {
}
```

Der Aufruf einer solchen parametrisierten Objektmethode erfolgt analog zur parameterlosen Methode, nur mit den gültigen Argumenten.

```
meinObject.meineParametrisierteMethode(0 , "Hallo");
```

Methoden können nur innerhalb einer Klasse definiert werden und haben immer einen Rückgabewert. Dies ist ein wenig verwirrend, aber wenn Sie wissen, dass `void` ein Rückgabewert ist, ist dies schnell klar. Die `void`-Methode ist die einfachste Methode in Java. Jede

Methode, die einen Wert zurückgibt, außer `void`, muss das `return`-Statement enthalten. Die `return`-Anweisung liefert den Wert zurück und **kann** auch in `void`-Methoden vorkommen.

Der Datentyp des Rückgabewertes einer Methode wird im Kopf der Methoden bei der Deklaration festgelegt.

```
void hallo() {  
}
```

Rückgabetyp: `void`, `void` wird in Java syntaktisch wie ein Datentyp behandelt, darf aber nur in Methoden genutzt werden.

```
int getGeschwindigkeit(){  
    return 0;  
}
```

Rückgabetyp: `integer`

Rückgabewert: 0

```
void getName(){  
    String name = "Peter";  
    return name;  
}
```

Rückgabetyp: `String`

Rückgabewert: Peter

Außerdem ist zu beachten, dass Code, der nach dem `return`-Statement steht, nicht mehr ausgeführt wird.

Listing 3.12 Methode mit nicht ausführbarem Code

```
void getName(){  
    String name = "Peter";  
    return name;  
    name = "Hanz"; //unerreichbarer Code  
}
```

Diese Methode `getName` liefert immer Peter zurück, der Code nach `return` wird nie ausgeführt.

3.3.6 Auf Eigenschaften von Objekten zugreifen

Eigenschaften (Attribute), auch Instanzvariablen genannt, werden in Java üblicherweise mit dem *Sichtbarkeitsmodifizierer* `private` deklariert. Dies bedeutet, auf die Attribute kann nur innerhalb der Klasse zugriffen werden. Der öffentliche Zugriff auf diese privaten Attribute erfolgt dann über so genannte Getter- und Setter-Methoden.

Listing 3.13 Getter- und Setter-Methoden einer Klasse

```
public class Motorrad {
    //Attribute
    private String typ;
    // Getter
    public String getTyp(){
        return typ;
    }
    //Setter
    public void setTyp(String typ){
        this.typ = typ;
    }
}
```

Der Vorteil dieses Zugriffs ist, dass Objekteigenschaften nicht einfach verändert werden und Sie bei der Zuweisung zusätzliche Prüfungen durchführen können.

3.3.7 Abstrakte Klassen und Methoden

Am Anfang haben wir bereits erläutert, dass Klassen **Baupläne** für Objekte sind. Man kann also vereinfacht sagen, abstrakte Klassen sind Baupläne für Unterklassen, eine Unterklasse erbt alle Eigenschaften (Attribute) und Methoden (Verhalten) einer abstrakten Klasse. Was soll das jetzt, denken Sie weiter, wir haben Autos, Motorräder und auch LKW. Alle drei sind Kraftfahrzeuge, alle Kraftfahrzeuge können bremsen und beschleunigen. Da macht es Sinn, eine Oberklasse zu schaffen, die dieses Verhalten implementiert, die Klasse Kfz. Sollten Sie zu einem späteren Zeitpunkt die Klasse Lkw benötigen, lassen Sie die neue Klasse Lkw von Kfz erben und haben somit einen kompletten Bauplan für die neue Klasse, ohne erst überlegen zu müssen, welches Verhalten Sie einheitlich den anderen Kraftfahrzeugen gegeben haben. Eine abstrakte Klasse wird durch das Schlüsselwort **abstract** gekennzeichnet. Das Verhalten eines Objektes kann in der konkreten Klasse überschrieben werden, wenn das konkrete Verhalten des Objektes nicht dem Verhalten der Oberklasse entspricht.

Listing 3.14 Abstrakte Klasse Kfz

```
public abstract class Kfz{
    private int geschwindigkeit;
    public void beschleunigen(){
        geschwindigkeit = geschwindigkeit + 1;
    }

    public void bremsen(){
        geschwindigkeit = geschwindigkeit -1;
    }
    public int getGeschwindigkeit(){
        return this.geschwindigkeit;
    }
}
```

Um eine Unterklasse von einer abstrakten Klasse zu erstellen, müssen Sie in der Klassendeklaration festlegen, welche abstrakte Klasse erweitert wird. Das Ableiten einer abstrakten Klasse erfolgt durch das Schlüsselwort **extends** (erweitern).

Listing 3.15 Unterkelas Auto der abstrakten Klasse Kfz

```
public class Auto extends Kfz {
}
```

Diese Unterkelas Auto ist voll funktionsfähig und erbt ihr Verhalten von der Oberklasse Kfz, da die Klasse Auto die Klasse Kfz erweitert, kann Auto natürlich eigenes Verhalten hinzufügen. Auto kann bremsen, beschleunigen und die Geschwindigkeit ausgeben. Zu beachten ist, dass nur öffentliche Methoden, also Methoden, die als public deklariert sind, vererbt werden können.

Jetzt fehlen nur noch die abstrakten Methoden einer abstrakten Klasse. Abstrakte Methoden innerhalb einer Klasse werden ebenfalls mit dem Schlüsselwort abstract deklariert. Mit abstrakten Methoden deklarieren Sie nur, welches Verhalten die Unterklassen implementieren müssen. Am Beispiel eines Autos ist dies wieder einfach zu erklären. Denken Sie jetzt an den Startvorgang Ihres Autos. Die meisten Autos werden noch durch Drehen des Zündschlüssels gestartet. Nun denken Sie darüber nach, ob dies bei allen Fahrzeugen so ist. Natürlich nicht. Manche Fahrzeuge haben einen Startknopf, ältere Fahrzeuge oder mit Diesel betriebene sowie LKW müssen zuerst vorglühen und so weiter. Wir halten nur fest, alle Fahrzeuge werden auf die eine oder andere Weise gestartet, auch Elektrofahrzeuge. Das konkrete Verhalten ist aber so unterschiedlich, dass es nicht möglich ist, dieses Verhalten für alle Fahrzeuge gleich festzulegen, und hier kommen die abstrakten Methoden ins Spiel. Jede Unterkelas von Kfz muss die Methode starten implementieren, sich aber selber darum kümmern, wie dies erfolgt. Eine abstrakte Methode deklarieren Sie wie folgt: *public abstract Rückgabetyp Bezeichner();*

Listing 3.16 Klasse Kfz mit abstrakter Methode „starten“

```
public abstract class Kfz{
public abstract void starten();
}
```

In Listing 3.17 sehen Sie, wie die abstrakte Methode starten in der Unterkelas Motorrad überschrieben werden könnte.

Listing 3.17 Klasse Motorrad mit überschriebener Methode „starten“

```
public class Motorrad extends Kfz{
@Override
    public void starten() {
        boolean gestartet = false;
        if(gestartet != true ){
            oeffnenBezinhahn();
            anZuendung();
            ...
        }
    }
}
```



HINWEIS: Unterklassen erben alle Methoden, die public deklariert sind.
Konstruktoren werden nicht vererbt.

3.3.8 Interfaces (Schnittstellen)

Sie wissen nun, wie man mit Objekten und Klassen umgeht und haben auch in der Test-App ein Interface benutzt. Doch: Was sind Interfaces und wozu benötigt man diese?

Wir wollen dies nun kurz erläutern. Interfaces haben den Zweck, öffentlich sichtbare Methoden ohne konkrete Implementierung einheitlich bereitzustellen. Sie haben in Ihrer ersten App schon das Interface `OnClickListener` genutzt, um mit einem Klick auf den Button zu reagieren. Mithilfe von Interfaces werden Funktionalitäten (Verhalten) von Klassen angeboten, die gleiche Funktionalitäten bieten. Dies bedeutet, in einem Interface werden nicht alle Methoden einer Klasse, sondern gleiche Methoden mehrerer Klassen einheitlich angeboten. Jede Klasse, die das Interface implementiert, muss auch diese Methoden anbieten. Das Interface bietet aber nur die Methode an, die eigentliche Funktionalität wird in der Klasse definiert. Daher sind Interfaces *abstract*. Wozu das Ganze? Wir erstellen ein Interface-Gewicht, das die Methoden `setGewicht` und `getGewicht` deklariert. Schauen Sie ins wirkliche Leben, wenn Sie verreisen, hat Ihr Koffer ein Gewicht, im Straßenverkehr hat ein Fahrzeug ein Gewicht. Alle diese Gewichte sind unterschiedlich und können sich auch verändern. Denken Sie an das wirkliche Leben, das Gewicht eines LKW ist abhängig davon, ob er beladen ist und was er geladen hat. Ebenso Ihr Auto. Fahren Sie mit Gepäck oder ohne. Oder der Koffer, ist er gepackt oder leer und so weiter. Jetzt haben wir nicht nur Klassen, die von `Kfz` erben, sondern auch noch einen Koffer, der ja nun wirklich nichts mit Kraftfahrzeugen zu tun hat.

Listing 3.18 Interface-Gewicht

```
public interface Gewicht {  
    public abstract void setGewicht(int gewicht);  
    public abstract int getGewicht();  
}
```

Implementiert eine Klasse dieses Interface, so muss diese Klasse die beiden Methoden `setGewicht` und `getGewicht` implementieren. In welche Klasse, Ober- oder Unterklasse, Sie dieses Interface implementieren, ist davon abhängig, ob es Sinn macht, dass alle Unterklassen diese Methoden erben. Benötigen wenige oder sogar nur eine Unterklasse dieses Interface, sollten Sie das Interface in die konkreten Unterklassen implementieren.

Listing 3.19 Klasse Motorrad mit Interface-Gewicht

```
public class Motorrad extends Kfz implements Gewicht {  
    int gewicht;  
    @Override  
    public void setGewicht(int gewicht){  
        this.gewicht = gewicht;  
    }  
  
    @Override  
    public int getGewicht(){  
        return this.gewicht;  
    }  
}
```

3.3.9 Java Packages (Pakete)

Der Java-Sourcecode wird logisch in Paketen organisiert. Diese Organisation in Paketen bietet sehr viele Vorteile. Klassen werden in eigenen Dateien gespeichert. Somit kann ein großes Programm sehr viele Klassendateien enthalten und schnell unübersichtlich werden. Durch die logische Organisation in Paketen behält man besser die Übersicht, außerdem können Klassen gleiche Namen besitzen, jedoch müssen sich diese gleichnamigen Klassen in unterschiedlichen Paketen befinden. Packages werden auf der Festplatte als Ordner abgelegt. Alle Klassendateien eines Pakets finden Sie dann in diesem Ordner. Java referenziert Klassen immer komplett mit dem Paketnamen. Deshalb ist es möglich, gleiche Klassennamen zu verwenden. Dieser gesamte Name, bestehend aus Paket- und Klassennamen, wird als Full Qualified Domain Name = **FQDN** bezeichnet.

3.3.10 Logging

Java bietet mit der Klasse `java.util.logging` ein gutes Werkzeug, um Informationen von Java-Programmen zur Laufzeit zu erhalten. `java.util.logging` ist auch in Android verfügbar. Der Einsatz von `java.util.logging` kann unter Umständen sinnvoll sein, allerdings werden wir nicht darauf eingehen.

Android bietet ebenfalls mit den Klassen `android.util.log` und `Logcat` ein sehr einfach zu benutzendes Werkzeug, um Programmmeldungen zur Laufzeit auszugeben. Daher beschränken wir uns auf die Benutzung von `android.util.log`. Der Einsatz dieser Klasse ist denkbar einfach. Sie müssen nur an der entsprechenden Stelle im Quellcode zum Beispiel `Log.d(TAG, "Logausgabe");` aufrufen und das Paket `android.util.log` importieren. Sie erhalten zur Laufzeit die Ausgabe dann in der `Logcat View` angezeigt. Die Logging-Klasse von Android bietet verschiedene Ausgabe-Level an.

DEBUG	<code>Log.d()</code>
ERROR	<code>Log.e()</code>
INFO	<code>Log.i()</code>
VERBOSE	<code>Log.v()</code>
WARN	<code>Log.w()</code>

3.3.11 Kommentare

Sie sollten grundsätzlich nicht mit Kommentaren sparen. Wenn Sie nach einiger Zeit den Programmcode wieder erweitern möchten oder auftretende Fehler analysieren müssen, sind Kommentare sehr hilfreich und sparen Zeit. Üblicherweise werden Kommentare in Java in Englisch verfasst, aber ein deutscher oder spanischer Kommentar ist allemal besser als gar kein Kommentar. Java bietet die Möglichkeit, Kommentare auf zwei Arten im Quellcode abzulegen.

Möglichkeit 1: Ein Kommentar wird mit // gekennzeichnet. Alles was hinter den Schrägstrichen steht, wird vom Compiler bis zum Ende der aktuellen Zeile ignoriert. Jede weitere Kommentarzeile muss wieder mit // beginnen.

Beispiel:

```
public void getValue() {  
    int i = 1; //Dies ist ein Kommentar!  
    // noch ein Kommentar  
}
```

Möglichkeit 2: Ein Kommentar wird mit /* begonnen. Dieser Kommentar muss wieder mit */ geschlossen werden. Alle Zeichen bis zum Abschluss ignoriert der Compiler. Mit dieser Art können Sie auch einen Kommentar mitten im Code erstellen.

Beispiel 1:

```
public void getValue(){  
/* Dies  
* ist  
* ein  
* mehrzeiliger Kommentar  
*/  
}
```

Beispiel 2:

```
public void getValue(){  
String name /* Kommentar im Quellcode */ = "Peter";  
}
```

Sie sollten aber noch den Code erkennen und nicht in den Kommentaren suchen müssen.

3.3.12 Ablaufsteuerung mit einfacher Verzweigung

Einfache Verzweigungen oder Bedingungen werden, wie in anderen Programmiersprachen auch, mit if-Abfragen durchgeführt.

Listing 3.20 if/else-Verzweigungen

```
int zahl = 5;  
  
if (zahl < 5){  
    Log.d("Logging","Zahl ist kleiner 5");  
}  
else if (zahl == 5){  
    Log.d("Logging","Zahl ist gleich 5");  
}  
else{  
    Log.d("Logging","Zahl ist größer 5");  
}
```

3.3.13 Ablaufsteuerung mit Mehrfachverzweigungen

Ist es jedoch notwendig, eine Bedingung auf mehrere verschiedene Werte zu überprüfen, wird dies mit *if/else* sehr schnell unübersichtlich und fehleranfällig. Listing 3.21 soll nur die Syntax von *switch/case* erläutern. Die *switch*-Anweisung erlaubt die Verwendung aller primitiven Datentypen, außer *Long* und *String*. *Enums* werden ebenfalls unterstützt.

Listing 3.21 switch/case-Verzweigungen

```
int zahl = 5;
switch(zahl){
    case 1:
    case 3:
    case 5:
        Log.i("Logging", "Zahl ist ungerade");
        break;
    case 2:
    case 4:
        Log.i("Logging", "Zahl ist gerade");
        break;
    default:
        Log.i("Logging", "keine gültige Zahl");
        break;
}
```

3.3.14 Wiederholungen mit Schleifen

Schleifen werden eingesetzt, um Anweisungen mehrfach auszuführen, ohne den Code der Anweisung/en mehrfach schreiben zu müssen. In Java gibt es drei Arten von Schleifen:

- die Zählschleife (for)
- die im Kopf prüfende Schleife (while/do)
- die im Fuß prüfende Schleife (do/while)

Zur Verdeutlichung wird ein einfaches String-Array mit allen drei Schleifentypen durchlaufen.

Listing 3.22 Mit der Zählschleife (for) ein Array durchlaufen

```
String hersteller[];
hersteller = new String[4] ;
hersteller[0] = "BMW";
hersteller[1] = "Honda";
hersteller[2] = "Suzuki";
hersteller[3] = "Yamaha";

for(int i=0; i < hersteller.length; i++){
    Log.d("Hersteller", hersteller[i]);
}
```

Listing 3.23 Mit der im Kopf prüfenden Schleife (while/do) ein Array durchlaufen

```
String hersteller[];
hersteller = new String[4] ;
hersteller[0] = "BMW";
hersteller[1] = "Honda";
hersteller[2] = "Suzuki";
hersteller[3] = "Yamaha";

int i=0;
while(i < hersteller.length){
    Log.d("Hersteller", hersteller[i]);
    i++; //Zähler erhöhen
}
```

Listing 3.24 Mit der im Fuß prüfenden Schleife (do/while) ein Array durchlaufen

```
//eine weitere Möglichkeit ein Array zu erstellen
String hersteller[] = {"BMW", "Honda", "Suzuki", "Yamaha"};
int i=0;
do{
    Log.d("Hersteller", hersteller[i]);
    i++; //Zähler erhöhen
} while(i < hersteller.length);
```

Alle drei Schleifen bringen in diesem Beispiel das gleiche Ergebnis, jedoch ist der Ablauf unterschiedlich. In Java gibt es *keine foreach Schleife* wie in anderen Programmiersprachen, jedoch bietet die *for Schleife* mit Iteration einen vollständigen Ersatz.

Listing 3.25 for Schleife mit Iteration

```
String hersteller[] = {"BMW", "Honda", "Suzuki", "Yamaha"};

for(String item: hersteller){
    Log.d("Hersteller: " +item);
}
```

3.3.15 Fehlerbehandlung

In Java gibt es zur Fehlerbehandlung, wie in vielen anderen Programmiersprachen auch, die Möglichkeit, einen *try/catch*-Block zu verwenden. Wobei die Bezeichnung Fehlerbehandlung nicht ganz korrekt ist. *Exception* bedeutet *Ausnahme*. Die Übersetzung von Try und Catch lautet ja *versuchen* und *fangen*. Das bedeutet, dass versucht wird, Code in einem *try*-Block auszuführen, und wenn dies nicht erfolgreich ist, fängt der *catch*-Block den Fehler. Daher kommt die umgangssprachliche Bezeichnung *Fehler abfangen*.

In Listing 3.26 sehen Sie ein einfaches Beispiel für eine Fehlerbehandlung. Was passiert? Es wird eine Division 4:0 durchgeführt, und das Ergebnis soll der Variablen z zugewiesen werden. Da aber eine Division durch 0 einen Fehler verursacht, würde die Anwendung ohne Fehlerbehandlung abstürzen. Da $z = 4 \% 0$ in einem *try*-Block eingeschlossen ist, läuft Ihr Programm weiter, und es wird im *Logcat* die Meldung Fehler mit einer Fehlernachricht ausgegeben.

Listing 3.26 Einfaches Beispiel zur Fehlerbehandlung mit try und catch

```
int z;  
try{  
    z = 4 % 0;  
}catch (Exception e){  
    Log.d("Fehler", e.getMessage());  
}
```

Dies ist natürlich nur ein allgemeines Beispiel, da im catch-Block keine Fehlerbehandlung durchgeführt und auch nur die ganz allgemeine Klasse *Exception* verwendet wird.



PRAXISTIPP: Exceptions (Ausnahmen) sollen immer dort behandelt werden, wo sie auch gefangen werden können.



HINWEIS: In den folgenden Kapiteln wird aus Gründen der Übersichtlichkeit und um den Code-Umfang zu reduzieren, auf eine Fehlerbehandlung meistens verzichtet. In Ihren eigenen Projekten sollten Sie, sofern notwendig, stets eine Fehlerbehandlung nach dem vorgestellten Schema verwenden.

4

Grundlagen von Layouts, Views, Komponenten und Intents: Erste Oberflächen erstellen

In Kapitel 3 haben Sie bereits eine erste kleine App erstellt und einen Schnelleinstieg in Java und Eclipse erhalten. Ab diesem Kapitel werden Sie schrittweise durch die Erstellung der ausgewählten Beispiel-App dieses Buches geführt. In jedem Kapitel werden dabei zunächst die notwendigen Grundlagen erklärt. In diesem Kapitel erläutern wir die Grundlagen von Layouts, Views, Komponenten und Intents einer Android-Anwendung.



Die Beispieldateien zu diesem Kapitel finden Sie unter
www.downloads.hanser.de im Unterordner *scytenotes 0.1*

■ 4.1 Zielsetzung

Am Ende dieses Kapitels werden Sie eine erste Version der Notizen-App mit den vier Haupt-Activities, den Grundnavigationselementen und der Navigation erstellt haben. Diesen Zwischenstand werden Sie kompilieren und erstmals testen können. Weiterhin werden Sie Layouts, Views, Ereignismethoden und Intents kennenlernen. Sie werden nun auf Klick-Ereignisse der Oberfläche reagieren und weitere Activities durch Intents starten können.

■ 4.2 Layouts, Views und Komponenten

Um ein Verständnis für Android-Anwendungen zu erhalten, benötigen Sie einige Grundlagen. Wir wollen Ihnen in diesem Abschnitt die notwendigen Basisinformationen vermitteln. Außerdem hat Google mittlerweile einen DesignGuide veröffentlicht, der Empfehlungen und UI-Prinzipien für Oberflächen unter Android bereitstellt. Dieser DesignGuide ist eine unverzichtbare Quelle für die Erstellung von homogenen Oberflächen. Sie finden ihn unter <http://developer.android.com/design/index.html>.

4.2.1 Layouts

Jede Android-Anwendung besteht aus mindestens einer oder mehreren Activities und Layouts. Layouts enthalten wiederum die View-Elemente der Oberfläche. Ein Layout kann man also als Rahmen mit speziellen Eigenschaften für Ausrichtung und Anordnung von View-Elementen auf der Oberfläche definieren. Ein View-Element ist wiederum ein einzelnes Element der Oberfläche. Es gibt in Android keine Klasse *Layout*, jeder einzelne Layout-Typ ist eine eigene Klasse, die die Klasse *ViewGroup* erweitert. Layouts können allerdings nicht nur View-Elemente aufnehmen, sondern auch wieder Layouts. Damit ist eine Verschachtelung der Oberflächenelemente möglich.

Jede Layout-Datei ist nach einem festen Schema aufgebaut und muss folgende Einträge enthalten:

1. den Name-Space:

```
xmlns:android=http://schemas.android.com/apk/res/android
```

Der Name-Space (Namensraum) ist für die Bereitstellung der Attribute für Layouts verantwortlich. Im Standard werden die Layout-Attribute von Android genutzt. Sollten Sie in Zukunft Fremdbibliotheken einsetzen, müssen Sie unter Umständen den Namensraum erweitern oder komplett ändern.

2. die Höhe und Breite:

- `android:layout_width="match_parent"`
- `android:layout_height="match_parent"`
 - `match_parent`: Die Größe des Layouts wird an die übergeordnete Größe angepasst.
 - `fill_parent`: Das übergeordnete Element wird ausgefüllt.
 - `wrap_content`: Die Größe wird durch den darzustellenden Inhalt bestimmt.

Android stellt im Standard eine Reihe von verschiedenen Layouts zur Verfügung. Diese sind:

- **LinearLayout:** Im LinearLayout werden alle View-Elemente, je nach Orientierung, Element für Element untereinander oder nebeneinander aneinander gereiht dargestellt. Sie sollten eine sehr intensive Nutzung und Verschachtelung von LinearLayouts in einer Oberfläche aus Performance-Gründen vermeiden und versuchen, andere Layout-Typen einzusetzen. Häufig kann ein tief verschachteltes LinearLayout durch ein TableLayout oder GridLayout ersetzt und vereinfacht werden.
- **TableLayout:** Das TableLayout bietet die Möglichkeit, gleichförmig zusammenhängende Oberflächen spalten- und zeilenbasiert zu erstellen. Ein sehr häufiger Einsatz des TableLayouts sind Eingabeformulare. Im TableLayout haben Sie die Möglichkeit, unterschiedliche Zeilenhöhen und Spaltenbreiten anzugeben, allerdings orientiert sich das TableLayout immer an der Zelle mit der größten Breite.
- **GridLayout:** Das GridLayout ist ähnlich dem TableLayout, kann jedoch so einfach wie das LinearLayout angewendet werden. Das GridLayout orientiert sich wie das TableLayout bei der Ausrichtung an Zeilen und Spalten. Die Besonderheit ist, dass mit dem GridLayout sehr effizient Grafikelemente positioniert werden können, die eine unterschiedliche Größe besitzen, und dass die Größe jeder einzelnen Zelle dynamisch angepasst wird. Um diesen Effekt zu erreichen, definieren Sie keine feste Zeilen- oder Spaltenposition und

benutzen das Attribut `rowSpan`. Außerdem müssen die einzelnen View-Elemente in der Größe proportional zueinander und die Form gleich sein.

- **FrameLayout:** Das FrameLayout ist die einfachste Art, aber auch das performanteste Layout. Alle Layout-Elemente werden in der linken oberen Ecke ausgerichtet und übereinander gestapelt. Dies bietet die Möglichkeit, durch intelligentes Verschachteln von Layout-Elementen, Elemente zu erstellen, die eine Hintergrundgrafik und einen Text im Vordergrund darstellen.
- **RelativeLayout:** In diesem Layout werden alle View-Elemente relativ zueinander positioniert. Die Ausrichtung eines View-Elements erfolgt am äußeren Rahmen, der vertikalen bzw. der horizontalen Mitte oder dem vorhergehenden View-Element.
- **AbsoluteLayout:** Dieser Layout-Typ wurde als *deprecated* (überholt) gekennzeichnet, und Sie sollten das Layout nicht mehr einsetzen. Benutzen Sie stattdessen das *RelativeLayout*.

Density-independent pixel

Bei der Layout-Erstellung wird die Einheit für Abstände oder Größen *density-independent pixel* (dp) verwendet. Die *density-independent pixel* sind eine abstrakte Einheit, um eine einheitliche Darstellung auf verschiedenen Display-Größen und -Auflösungen zu gewährleisten. Die Berechnung der *dp* erfolgt auf der Basis der Pixeldichte des Bildschirms, relativ zu 160 dpi (160 dpi entspricht 1 Zoll Bildschirmauflösung). Die Formel zur Umrechnung ist sehr einfach:

$$\text{pixels} = \text{dps} * (\text{density} / 160)$$

4.2.2 Views und Widgets

Genau genommen sind fast alle Oberflächenelemente von Android-Views auch Layouts. Layouts erben von der Klasse `ViewGroup`, die wiederum die Klasse `View` erweitert, aber dies nur am Rande. Die meisten Tutorials und auch wir sprechen von Views, wenn wir ein einzelnes Oberflächenelement bezeichnen. Dies korrespondiert auch mit der Arbeitsweise des ADT. Die Bezeichnung Widget hingegen wird bei Android unterschiedlich genutzt und sorgt manchmal für Verwirrung. Als Widgets werden meist komplexere Elemente bezeichnet, die mehr Funktionalität oder Animation bieten, wie z.B. DatePicker oder TimePicker. Die Android-API-Referenz listet im Paket `android.view` alle abstrakten Klassen und Hilfsklassen auf. Im Paket `android.widget` befinden sich die eigentlichen Benutzer-Schnittstellen-Elemente. Aus Platzgründen werden wir hier nicht auf die View-Elemente eingehen. Sie werden im weiteren Verlauf dieses Buches die wichtigsten Elemente und deren Verwendung kennenlernen.

4.2.3 Bausteine einer Android-Anwendung

Jede Android-Anwendung besteht aus einem oder mehreren Bausteinen. Eine Activity ist ein solcher Baustein oder auch eine solche Basiskomponente. Jede Basiskomponente kann mehrfach in einer Android-Anwendung vorkommen. In diesem Abschnitt stellen wir die Basisbausteine einer Android-Anwendung vor und erläutern kurz deren Einsatzzweck.

Alle Basiskomponenten haben einen Lebenszyklus (Lifecycle). Diesen Lifecycle gilt es bei der Programmierung zu beachten. Abhängig vom Lifecycle der Komponente entscheidet sich, ob Daten oder Funktionen zur Laufzeit zur Verfügung stehen, oder ob diese verloren gehen.

Um sich einen Überblick zu verschaffen, welche Komponenten in einer Android-Applikation eingesetzt werden können, öffnen Sie zunächst in Eclipse die Datei *AndroidManifest.xml*, TAB APPLICATION und dann im Bereich *Application Nodes* den Dialog zum Hinzufügen eines Eintrags durch Klick auf ADD.... Jedoch sind nicht alle angezeigten Einträge wirkliche Komponenten. Einige Elemente der Übersicht werden benötigt, um zusätzliche Informationen oder Verweise zu deklarieren.

Basiskomponenten

- **Activity:** Activities sind die Anwendungskomponenten. In anderen Programmiersprachen nennt man diese Codebehind. Dies wäre in Android aber nicht ganz korrekt. Vereinfacht ausgedrückt ist jede Activity eine einzelne Bildschirmseite, die für die Darstellungslogik verantwortlich ist und die die Nutzereingaben verarbeitet. Activities werden im Projekt im Source-Ordner /src als eigene Klasse deklariert.

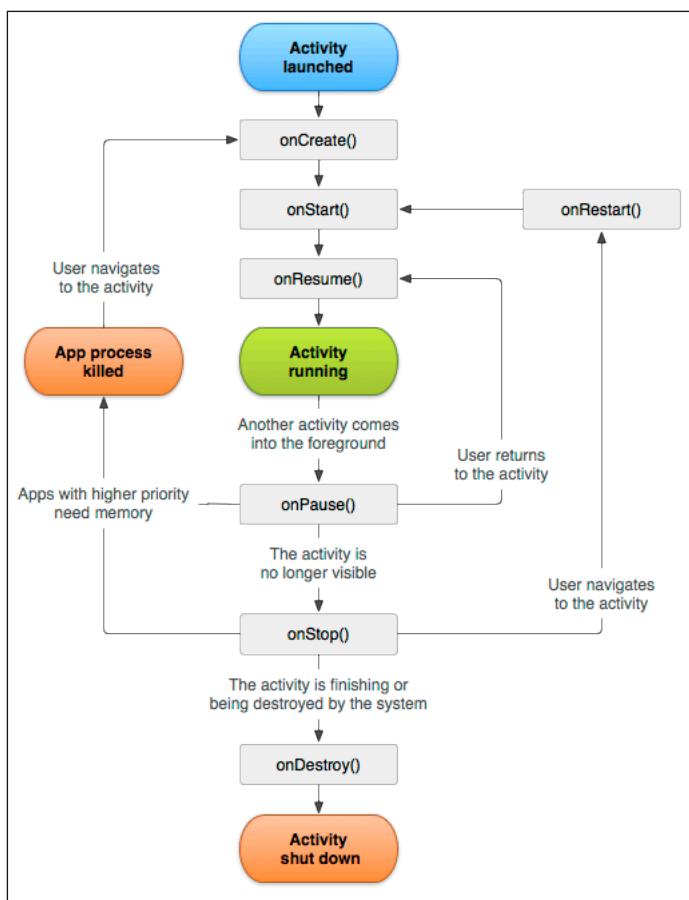


Bild 4.1
Lifecycle der Activity

- Fragments:** Fragments können Sie als Teile einer Activity verstehen. Fragments bestehen, wie auch Activities, aus einer Java-Datei und einem Layout und haben einen Lebenszyklus (Lifecycle), der der Activity ähnlich ist. Jedoch benötigen Fragments immer eine Activity, die sie umgibt und in deren Lifecycle sich Fragments integrieren. Der Nutzen von Fragments liegt darin, dass Teile eines Layouts mit Code in kleinere und wiederverwendbare Einheiten ausgliedert und somit schneller und effizienter Anwendungen für verschiedene Gerätetypen oder Displaymodi erstellt werden können. Sie können diese Fragmente in Layouts für verschiedene Auflösungen und Orientierungen des Bildschirms einbinden und wiederverwenden.

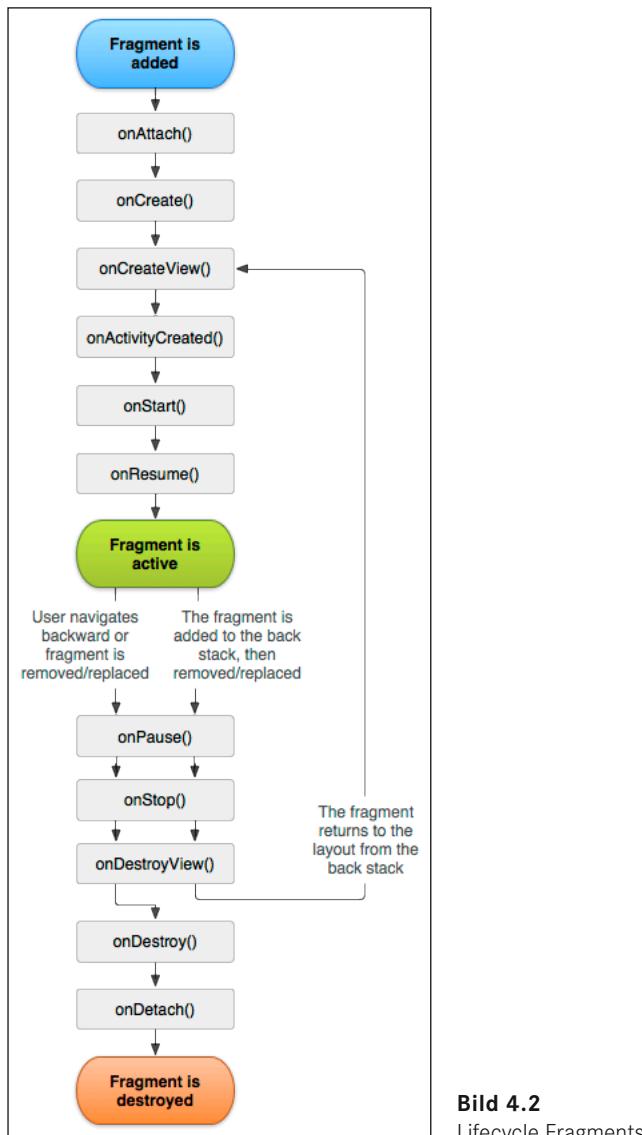


Bild 4.2
Lifecycle Fragments

- **Provider:** Provider bieten anderen Applikationen die Möglichkeit, auf Daten ihrer Anwendung zu zugreifen. Android stellt selbst schon einige Provider bereit, so genannte *Content Provider*. Wie der Name schon sagt, bieten diese Provider einen bestimmten Content (Inhalt) an, wie z.B. Kontakt- oder Kalenderdaten. Sinn und Zweck von Providern ist die Bereitstellung von bestimmten Inhalten, ohne anderen Anwendungen Zugriff auf alle Daten zu gewähren. Provider bieten eine REST-ähnliche Schnittstelle zur Abfrage von Daten an.

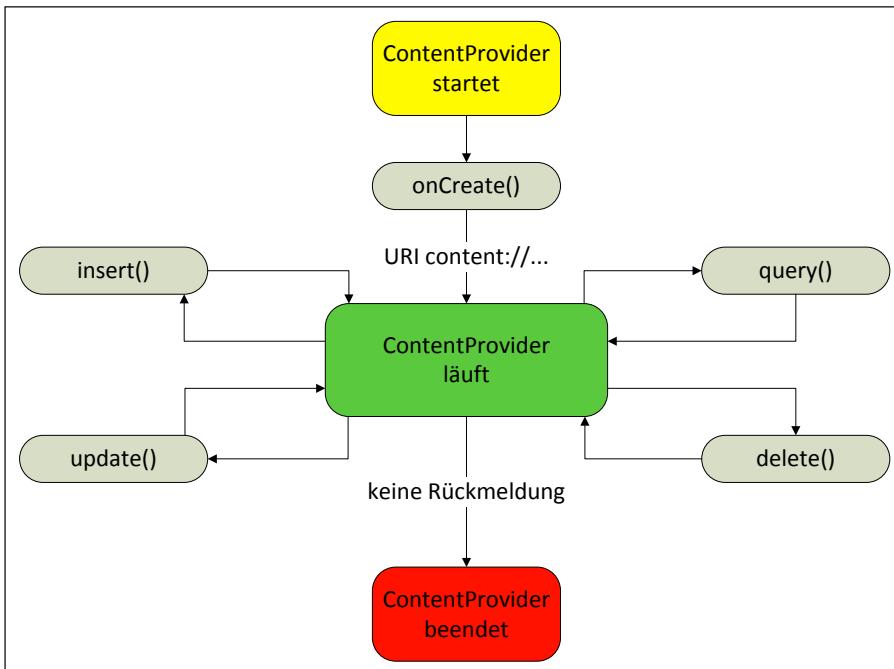


Bild 4.3 Lifecycle ContentProvider

- **Receiver:** Receiver sind Empfänger, wie der englische Name bereits verrät. Receiver empfangen Nachrichten, die vom System oder anderen Applikationen versendet werden, ohne dass diese einen speziellen Empfänger besitzen. Deshalb bezeichnet man Receiver auch als *BroadcastReceiver*. Denken Sie hierbei an Radiosender und Radioempfänger. Das System versendet permanent Nachrichten, z.B. über eingehende Anrufe, Ladezustand des Akkus, Status der Netzwerkverbindungen usw. Wollen Sie bestimmte Nachrichten empfangen und auswerten, so benötigen Sie dafür *BroadcastReceiver*.

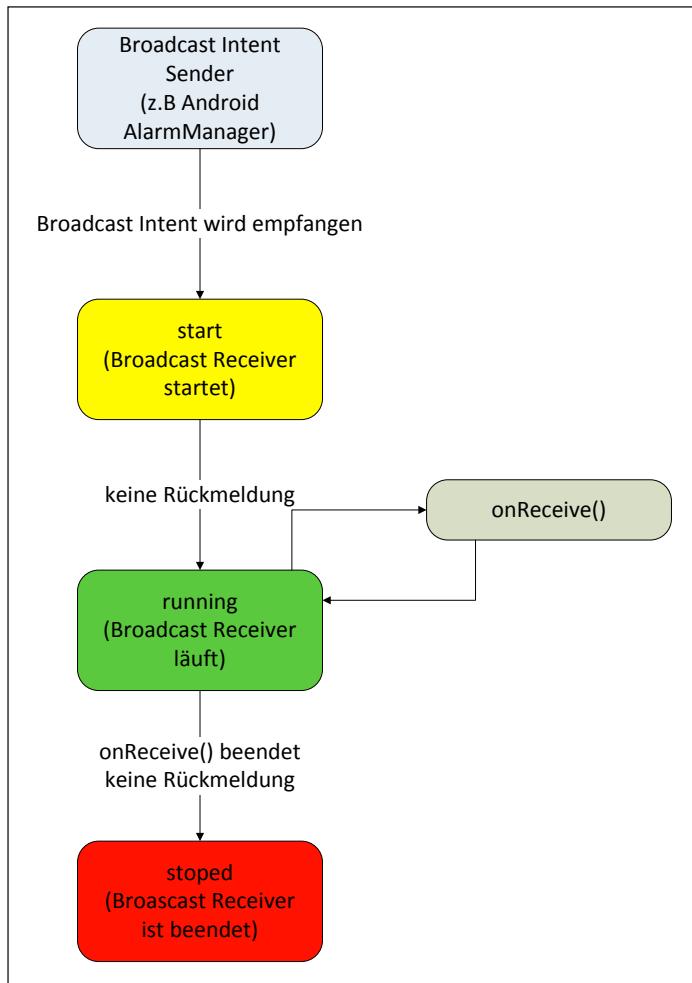


Bild 4.4
Lifecycle Broadcast
Receiver

- **Service:** Ein Service (Dienst) in Android erfüllt den gleichen Zweck wie ein Dienst in Windows oder ein Daemon in Linux. Services haben auch hier kein User-Interface und dienen zur Abarbeitung von Aufgaben im Hintergrund. Services können in zwei Formen auftreten:
 - **Started:** Ein Started-Service wird von einer anderen Anwendungskomponente gestartet, läuft eigenständig und beendet sich selbst nach Abarbeitung seiner Aufgabe.
 - **Bound:** Dieser Service wird an eine oder mehrere Anwendungskomponenten mit `bindService()` gebunden und bietet eine Client-Service-Schnittstelle zur Kommunikation. Der Service wird beendet, sobald keine Komponente mehr an ihn gebunden ist. Die Definition der Client-Service-Schnittstelle erfolgt in der *Android Interface Definition Language* (AIDL). Die AIDL ist nicht Bestandteil dieses Buches. Weitere Informationen hierzu finden Sie unter <http://developer.android.com/guide/components/aidl.html>.

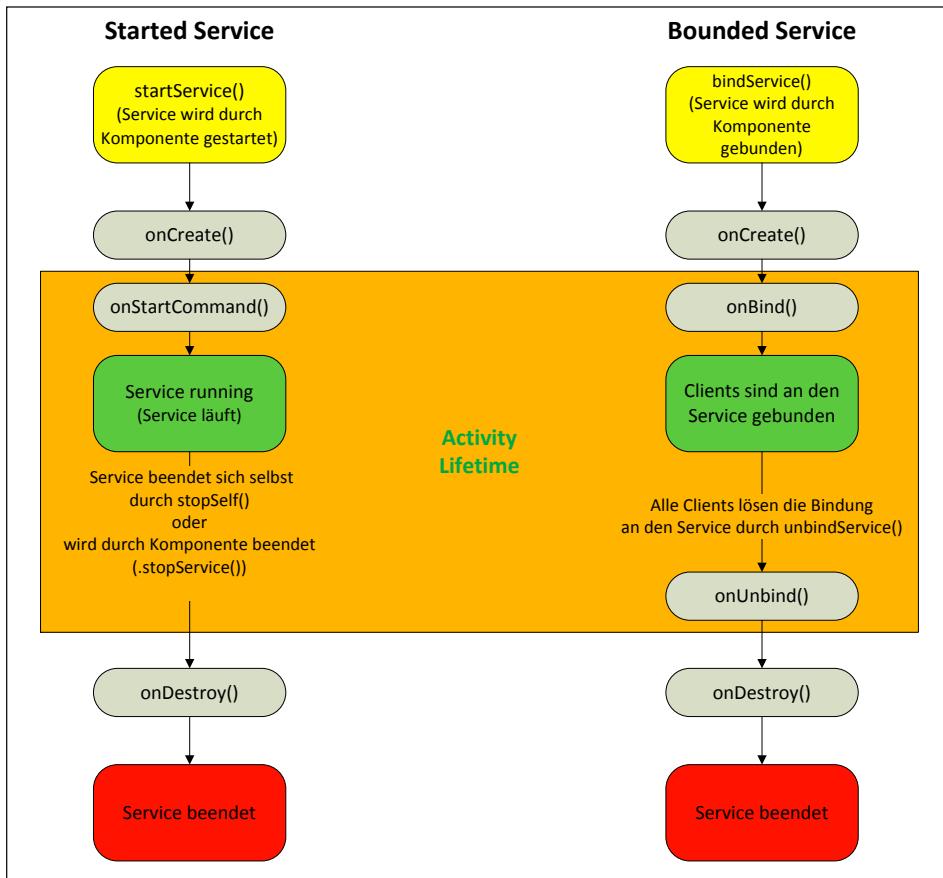


Bild 4.5 Lifecycle Service

Deklarationen

- **Activity-Alias:** Ein Activity-Alias ist, wie in anderen Systemen auch, ein symbolischer Name oder eine Parallelbezeichnung für die Activity. Damit wurde die Möglichkeit geschaffen, eine Activity ein weiteres Mal im Manifest zu deklarieren, um abweichende Einstellungen für diese vornehmen zu können, z. B. ein anderes Theme, ein eigenes Icon oder die Activity selbst mit anderen Parametern neu zu starten.
- **Meta Data:** Mit Metadaten werden zusätzliche Informationen im Manifest deklariert. Metadaten werden unter anderem für die Benutzung der GoogleMaps API benötigt. Metadaten werden im Android-Manifest als Key-Value-Paar deklariert und können für die Elemente `<activity>`, `<activity-alias>`, `<service>` und `<receiver>` angegeben werden.
- **Uses Library:** In diesem Tag deklarieren Sie gemeinsam genutzte Bibliotheken, mit denen die App verlinkt ist. Das System wird angewiesen, dass der Class-Loader diese automatisch beim Laden des Anwendungspakets laden soll. Weiterhin werden im Google Play Store alle Geräte herausgefiltert, die diese Bibliothek nicht bereitstellen.

■ 4.3 Layouts und Activities

Wir beginnen nun mit der Anlage eines neuen Projekts. In Abschnitt 3.1.1 haben wir dies bereits ausführlich erläutert. Bei der Erstellung des Projektes wählen Sie der Einfachheit halber folgende Einstellungen:

1. Assistent New Android Application:

- **Application Name:** scyte notes
- **Project Name:** scyNotes
- **Package Name:** eu.scyte.notes
- **Minimum Required SDK:** API 14: Android 4.0 (IceCreamSandwich)
- **Target SDK:** API 17 Android 4.2 (Jelly Bean)
- **Compile With:** API 17 Android 4.2 (Jelly Bean)
- **Theme:** Holo Light with Dark Action Bar

2. Assistent New Android Application, Configure Project:

- *Create Activity* = aktiviert
- *Create Project in Workspace* = aktiviert

3. Assistent New Android Application, Create Activity:

- *Create Activity* = aktiviert
- *Navigation Type* = Blank Activity

4. Assistent New Android Application, Blank Activity:

- **Activity Name:** MainActivity
- **Layout Name:** activity_main
- **Navigation Type:** None

Nachdem Sie das Projekt erfolgreich erstellt haben, werden wir in den nächsten Abschnitten die Anpassungen der Start-Activity (*MainActivity*) und des Layouts ausführlich erläutern. Am Ende dieses Kapitels können Sie die App auf Ihrem Gerät starten und navigieren.

4.3.1 Layouts erstellen

Die App enthält vier Haupt-Activities, um Notizen zu erfassen. Diese sind die Start-Activity (Main) zur Auflistung aller Notizen, die Text-Notiz-Activity zur Erfassung von Text, die Skizzen-Notiz zum Erstellen von Skizzen und Markierungen auf Bildern und die Voice-Notiz-Activity, um Sprach-Notizen aufzuzeichnen und wiederzugeben. Jede einzelne Activity benötigt ein eigenes Layout. Sie werden jetzt die Layouts und die notwendigen Navigationsmenüs erstellen. Wir werden dabei nicht auf alle Attribute von Layouts oder Views eingehen, denn dies würde den Rahmen dieses Buches sprengen. Anhand der Beispiel-App lernen Sie einige Einstellungen und deren Wirkung kennen. Das ADT liefert Ihnen auch zu jedem Attribut eine Kurzerklärung. Die Qualität dieser Hinweise müssen Sie selbst beurteilen.



HINWEIS: Nicht alle Bearbeitungsschritte lassen sich problemlos mit dem grafischen Editor lösen. Sie müssen häufig den XML-Code von Hand anpassen.

Layout der Start-Activity

1. Öffnen Sie die Datei *Layout activity_main.xml* im Ordner */res/layout*, falls diese noch nicht angezeigt wird. Entfernen Sie nun die im Layout angezeigte *TextView* „Hello World“.
2. Fügen Sie dem Layout nun eine *ListView* aus dem Bereich *Composite* hinzu. Stellen Sie folgende Attribute ein: *Id*, *Align Parent Left*, *Align Parent Top*, *Width* und *Height*.
3. Jetzt fügen Sie ein *TableLayout* aus dem Bereich *Layout* unterhalb der *ListView* ein. Das *TableLayout* enthält zwei Tabellenzeilen (*TableRow*).
4. In die erste *TableRow* fügen Sie nun drei *ImageButtons* ein.
5. In die zweite *TableRow* fügen Sie drei *TextView*-Elemente ein.

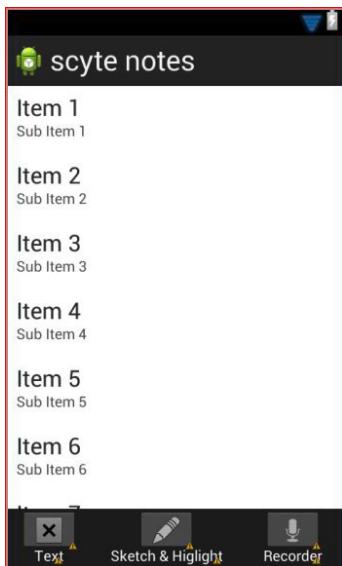


Bild 4.6
Layout der *activity_main.xml*

Listing 4.1 XML-Code *activity_main.xml*

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

    <ListView
        android:id="@+id/main_listView"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_above="@+id/main_bottomTableLayout"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" >

    </ListView>

    <TableLayout
        android:id="@+id/main_bottomTableLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:background="@android:drawable/dark_header"
        android:stretchColumns="0,1,2" >

        <TableRow
            android:id="@+id/tableRow1"
            >

            <ImageButton
                android:id="@+id/mainBottomActionBar_imageButton_textNote"
                android:layout_width="40dp"
                android:layout_height="40dp"
                android:layout_gravity="center"
                android:onClick="onClick"
                android:scaleType="center"
                android:src="@android:drawable/btn_dialog" />

            <ImageButton
                android:id="@+id/mainBottomActionBar_imageButton_scetchNote"
                android:layout_width="40dp"
                android:layout_height="40dp"
                android:scaleType="center"
                android:onClick="onClick"
                android:layout_gravity="center"
                android:src="@android:drawable/ic_menu_edit" />

            <ImageButton
                android:id="@+id/mainBottomActionBar_imageButton_voiceNote"
                android:layout_width="40dp"
                android:layout_height="40dp"
                android:scaleType="center"
                android:onClick="onClick"
                android:layout_gravity="center"
                android:src="@android:drawable/ic_btn_speak_now" />

        </TableRow>

        <TableRow
            android:id="@+id/tableRow2"
            android:layout_marginBottom="2dp"
            android:layout_marginTop="-4dp"
            >

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

```
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Text"
        android:textColor="@android:color/primary_text_dark" />

    <TextView
        android:id="@+id/mainBottomActionBar_textView_scetchNote"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Sketch &#38; Highlight"
        android:textColor="@android:color/primary_text_dark" />

    <TextView
        android:id="@+id/mainBottomActionBar_textView_voiceNote"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Recorder"
        android:textColor="@android:color/primary_text_dark" />
</TableRow>
</TableLayout>
</RelativeLayout>
```

Kurze Erläuterung des Aufbaus der activity_main.xml

- **RelativeLayout:** Dies ist der Layout-Container der Activity und präsentiert die gesamte Oberfläche. Im obersten Layout-Container muss immer der Namensraum angegeben werden. Da wir keine externen Layout-Bibliotheken nutzen, wird hier der Namensraum von Android angegeben (`xmlns:android=...`). Erst durch diese Angabe können die View-Elemente mit ihren Eigenschaften verwendet werden. Der zusätzliche Namensraum, der hier automatisch eingefügt wurde (`xmlns:tools=http://schemas.android.com/tools`), ist nötig, um die Designtools des ADT-Plug-ins vollständig zu nutzen.
- **ListView:** In der ListView werden später die einzelnen Notizen angezeigt. Durch die Attribute `android:layout_alignParentLeft="true"` und `android:layout_align Parent Top="true"` wird die ListView links oben am Layout ausgerichtet. Mit den Attributen `android:layout_width="match_parent"` und `android:layout_height="wrap_content"` legen Sie fest, dass die ListView sich an die Breite des übergeordneten Containers anpasst und nur so hoch ist, wie der anzugezeigte Inhalt. Das Attribut `android:layout_above="@+id/main_bottomTableLayout"` bewirkt, dass die ListView oberhalb des TableLayouts erstellt wird und darüber endet.
- **TableLayout:** Das TableLayout stellt den äußeren Container der Navigationsleiste dar. Die Navigationsleiste kennzeichnet sich durch drei Buttons, die bei einem Klick die entsprechende Unter-Aktivität aufrufen. Darunter wird ein Beschreibungstext angezeigt. Die *Id* des TableLayouts `android:id="@+id/main_bottomTableLayout"` wurde nur vergeben, damit der ListView mitgeteilt werden kann, über welchem Container sie sich positionieren soll. Mit der Eigenschaft `android:layout_width="match_parent"` legen wir fest, dass sich die Navigationsleiste über die gesamte Breite ausdehnen soll. Die Höhe der Tabelle soll nur so groß sein, wie der darzustellende Inhalt. Das wird mit `android:layout_height="wrap_content"` erreicht. Das Attribut `android:layout_`

`alignParentBottom="true"` bewirkt, dass die Tabelle am unteren Rand ausgerichtet ist. Mit `android:background="@android:drawable/dark_header"` wird die Hintergrundfarbe festgelegt. Sie können hier auch andere definierte Ressourcen angeben, wir haben die Farbe der *ActionBar* gewählt. Diese Farbressource ist im *Holo Theme* definiert. Damit die drei Spalten gleichmäßig aufgeteilt werden, müssen Sie im Attribut `android:stretchColumns="0,1,2"` den nullbasierten Index jeder Spalte festlegen.

- **TableRow1:** In der TableRow1 werden die Navigationsbuttons in einer Zeile dargestellt. Die Höhe und Breite haben wir an das 48-dp-Pattern aus dem DesignGuide angelehnt. In diesem Pattern gibt Google eine Empfehlung von Größenverhältnissen einer guten Oberfläche am Beispiel eines ListItems. Weitere Informationen finden Sie unter <http://developer.android.com/design/style/metrics-grids.html>.
- **ImageButton:** Wie die Höhe und Breite festgelegt werden, wissen Sie. Die noch unbekannten Attribute sind `android:onClick="onClick"`, hier wird der Methodename festgelegt, der bei einem Klick-Ereignis aufgerufen wird, `android:scaleType="center"`, dies ist die Art, wie die Image-Ressource dargestellt oder skaliert wird, center = zentriert, keine Skalierung, `android:layout_gravity="center"` für die Gewichtung, die festlegt, wie das Element in einer Gruppe von Zellen positioniert wird und `android:src="@android:drawable/ic_menu_edit"`, welches die Image-Ressource festlegt.
- **TableRow2:** Die TableRow2 enthält die Labels (TextView) der ImageButtons, um dem Anwender die Benutzung zu vereinfachen. Mit `android:layout_marginBottom="2dp"` `android:layout_marginTop="-4dp"` legen Sie den Abstand nach unten und oben fest.
- **TextView:** Das Attribut `android:text="Text"` legt den anzuzeigenden Text fest und wird gelb markiert, da dies ein fest programmierte String ist. Üblicherweise werden Strings als Ressource in der `strings.xml` im Ordner `/res/values` festgelegt.



PRAXISTIPP: Wenn Sie sich über die System-Ressourcen der Android-Version informieren möchten, so finden Sie diese im SDK im Ordner `/platforms/android-„Vers.Nr“/data/res`, sofern Sie die Quellen installiert haben.

Layout der Text-Notiz

1. Fügen Sie dem Layout der Text-Note, also der `activity_note_text.xml`, lediglich ein Multi-Line-`EditText`-Element aus dem Bereich `TextFields` hinzu.

Wie Sie im XML-Code erkennen (siehe Listing 4.1), wurde nicht jedem View-Element eine **Id** vergeben. Sie müssen einer View nur dann eine Id vergeben, wenn Sie diese programmatisch im Code verwenden wollen oder andere View-Elemente an dieser ausrichten wollen. Allerdings vergibt beziehungsweise verlangt das ADT-Plug-in bei der Erstellung einiger Elemente eine Id, z.B. bei Erstellung einer TableRow. Das Android-Ressource-System arbeitet intern mit Integer-Werten. Jedes View-Element, das eine Id besitzt, wird als statische Integer-Konstante mit einem automatisch generierten Wert in der Datei `R.java` angelegt.



Bild 4.7
Layout der Text-Notiz

4.3.2 Activities

In unserem Projekt verwenden wir nur Activities, die die Basisklasse der Activity erweitern. In Abschnitt 4.2.3 haben wir bereits erläutert, was eine Activity kennzeichnet. In Android sind aber auch noch andere Activity-Klassen verfügbar:

- **ListActivity:** Diese Activity ist speziell für die Darstellung von einfachen Listen geeignet und darauf optimiert.
- **NativeActivity:** Eine NativeActivity ist eine in nativem Code, also C oder C++ programmierte Activity. Die Android-Plattform macht es möglich, dass Activities einer App in verschiedenen Programmiersprachen erstellt werden können. Eine NativeActivity muss hierzu nur im Manifest deklariert werden, um sie nutzen zu können.
- **LauncherActivity:** Die LauncherActivity repräsentiert eine Liste von Activities, die über einen Intent gestartet werden können. In der Praxis werden diese Launcher eingesetzt, um Activities aufzulisten und zu starten, die für den gleichen Anwendungszweck entworfen sind. Sie können mit dieser Activity ganz einfach Activities, z. B. alle auf dem System installierten Browser, in einem übergeordneten Startbildschirm darstellen und starten.
- **ExpandableListActivity:** Diese Activity ist optimiert, um Listen darzustellen, die bei Auswahl eines Items expandieren, also eine weitere Auswahl oder Informationen eines Items innerhalb der Liste darstellen.

4.3.2.1 Eine neue Activity-Klasse erstellen

Am Beispiel der Text-Notiz zeigen wir Ihnen, wie Sie in Eclipse eine neue Activity-Klasse erstellen.

1. Erstellen Sie eine neue Klasse *NoteTextActivity*. Drücken Sie dazu die Tastenkombination ALT+SHIFT+N, wählen dann CLASS oder machen einen Rechtsklick auf den Paketnamen, wählen NEW und dann CLASS. Im folgenden Dialog vergeben Sie den Namen Ihrer neuen Klasse *NoteTextActivity* und geben die Klasse (*Superclass*) an, die erweitert werden soll. In unserem Fall ist dies die Klasse *Activity* von Android. Sie müssen jetzt nicht den kompletten Pfad der Oberklasse wissen, Eclipse unterstützt Sie hierbei. Tragen Sie in das Feld *Superclass* nur *Activity* ein und klicken dann auf *Browse....* Jetzt erhalten Sie eine Auflistung der Klassen, und *android.app* wird bereits richtig vorgeschlagen.
2. Fügen Sie nun in die Klasse die überschriebene Methode *onCreate(Bundle)* ein. Rufen Sie durch einen RECHTSKLIK und Auswahl von SOURCE oder durch Tastenkombination ALT+SHIFT+S das Auswahlfenster für die Source-Code-Bearbeitung auf und wählen OVERRIDE/IMPLEMENT METHODS. Im folgenden Fenster werden Ihnen alle abstrakten überschreibbaren Methoden der Klasse *Activity* aufgelistet. Navigieren Sie zu **onCreate (Bundle)**, markieren diese und bestätigen mit OK. Methoden, die schon in der Klasse implementiert sind, werden nicht angezeigt. Die Methode *onCreate(Bundle)* wird immer dann benötigt, wenn die Activity eine Layout-Ressource mit *setContentView(int)* definiert und mit *findViewById(int)* programmatisch auf View-Elemente des Layouts zugegriffen werden soll.
3. Nun weisen Sie der Activity das Layout zu. Erweitern Sie die Methode *onCreate(Bundle)* mit *setContentView(R.layout.activity_note_text)*. Mit diesem Aufruf legen Sie fest, welches Layout bei der Erstellung der Activity geladen wird.

Wiederholen Sie den Vorgang für die *NoteSketchActivity* und *NoteVoiceActivity*. Achten Sie bei der Zuweisung des Layouts darauf, dass Sie das richtige Layout wählen.



HINWEIS: Sollten Sie jetzt Ihre Activity starten und auf die Buttons klicken, würde die App abstürzen. Der Grund dafür ist, dass noch die Methode für das Klick-Ereignis fehlt.

■ 4.4 Ereignisse und Intents

Ein Ereignis (Event) ist nicht nur der Klick auf den Button. Das Laden eines Layouts ist z. B. ein Ereignis, auch Zustandsänderungen im System sind Ereignisse. Java stellt hierfür Ereignisklassen und ein Ereignismodell zur Verfügung. Die Entwickler von Android haben dieses Ereignismodell für die Oberflächenentwicklung stark erweitert und vereinfacht und somit ist es unter Android möglich, durch wenig Programmieraufwand auf Ereignisse der Oberfläche zu reagieren. Intents werden häufig durch Ereignisse der Oberfläche, also Benutzereingaben, versendet, das Starten einer Unter-Activity erfolgt durch einen solchen Intent. Deshalb folgen die Intents in diesem Buch nach den Ereignissen.

4.4.1 Auf Klick-Ereignisse der Oberfläche reagieren

Sie haben das XML-Layout der Activity *main* erstellt und wollen, dass bei einem Klick auf die ImageButtons die entsprechende Notiz-Ansicht geöffnet wird. Bevor Sie jedoch die Unter-Activities starten können, müssen Sie zuerst festlegen, welche View-Elemente ein Klick-Ereignis auslösen und was mit diesem Ereignis geschehen soll. Bei den ImageButtons im Layout der main-Activity (*activity_main.xml*) haben Sie dem Attribut `onClick` den Wert `onClick` zugewiesen (`android:onClick="onClick"`). Dieser Wert legt die Methode fest, die bei einem Klick-Ereignis im Code aufgerufen werden soll. Sie müssen die Methode in der Java-Klasse der Activity hinzufügen.

Code der MainActivity mit onClick-Methode und switch/case erweitern

1. Sie müssen zunächst die Methode `onClick(View v)` implementieren, um auf Klick-Ereignisse der ImageButtons zu reagieren. Der Methodenname `onClick` wurde durch die Zuweisung des Attributs `onClick` im XML-Layout festgelegt. Die Methode `onClick (View v)` wird ausgeführt, wenn auf einen *ImageButton* geklickt wird.

Listing 4.2 `onClick(View v)`-Methode

```
public void onClick(View v) {  
}
```

2. Um eine Unterscheidung treffen zu können, welcher Button geklickt wurde, wird die `onClick(View v)`-Methode um eine *switch/case*-Anweisung, die die *Id* der View auswertet, erweitert.

Listing 4.3 `onClick(View v)`-Methode mit *switch/case*

```
public void onClick(View v) {  
    switch (v.getId()) {  
        case R.id.mainBottomActionBar_imageButton_textNote:  
            break;  
        case R.id.mainBottomActionBar_imageButton_scetchNote:  
            break;  
        case R.id.mainBottomActionBar_imageButton_voiceNote:  
            break;  
        default:  
            break;  
    }  
}
```

4.4.2 Mit Intents eine andere Activity aus der aktuellen Activity aufrufen

Um eine andere Activity aus der aktuellen Activity aufzurufen, werden Intents benötigt. Was sind Intents? Stark vereinfacht ausgedrückt sind Intents Nachrichten bzw. Absichtserklärungen. Diese Absichtserklärungen oder Nachrichten werden in Android in Intents gekapselt. Intents werden unter anderem für die lose Koppelung von Activities eingesetzt. Sie können damit nicht nur eine weitere Activity starten, sondern auch Informationen über-

geben. Mit Intents versendet man also eine Absichtserklärung, welche Aktion ausgeführt werden soll, und gleichzeitig können Informationen mit versendet werden. Die in Abschnitt 4.2.3 angesprochenen BroadcastReceiver empfangen solche Intents. Es gibt zwei Arten von Intents, explizite und implizite.

Explizite Intents

Explizite Intents senden ihre Absicht an einen Empfänger, und der Empfänger muss explizit angegeben werden, daher explizite Intents. Empfänger können *Activities*, *Services* oder *BroadcastReceiver* sein, also Komponenten einer Android-Anwendung. Der Empfänger wird über den voll qualifizierten Klassennamen angesprochen. Befindet sich die Komponente in einem anderen Paket, so muss dies mit angegeben werden.

Beispiel:

```
Intent intent = new Intent(this, TestActivity.class);
intent.putExtra("message", "inhalt der message");
startActivity(intent);
```

Das Beispiel zeigt den Start der Klasse TestActivity durch einen Intent. Mit dem Intent werden Informationen (.putExtra(...)) an die TestActivity übergeben. In der TestActivity ist es dann möglich, die übergebenen Informationen auszuwerten. Mit der Methode .putExtra(...) des Intents können verschiedenste Informationen übergeben werden. Die Information werden als Key-Value-Paar übergeben. Der Key ist immer ein String und dient als Bezeichner, damit die Information vom Empfänger identifiziert werden kann. Der Value enthält die eigentlichen Daten. In diesem Beispiel ist der *Key = message* und *Value = Inhalt der message*. Der Value muss nicht zwingend immer ein String-Objekt sein, vielmehr kann Value unterschiedliche Datentypen annehmen. Somit können mit Intents unterschiedliche Informationen ausgetauscht werden.

Implizite Intents

Implizite Intents unterscheiden sich von expliziten darin, dass sie ihre Absicht absenden, aber der Empfänger nicht feststeht, oder vielmehr nicht klar ist, welche Komponente genau sich der Nachricht annimmt. Am einfachsten zu verstehen ist dies, wenn Sie an die *Send-to*- oder *Senden-an*-Funktion im Windows Explorer denken. Nach einem Rechtsklick auf eine Datei, wählen Sie *Senden an* und dann *E-Mail-Empfänger*. So startet automatisch das Standard-E-Mail-Programm Ihres Systems, und genau so verhalten sich implizite Intents. Sie erklären, was ihre Absicht ist, und das System entscheidet, welche Komponente sich darum kümmert. Der Vorteil in Android ist, dass im Standard schon eine Menge solcher Intents möglich sind. Sie können unter anderem einen Anruf starten, eine SMS oder E-Mail versenden, die Galerie aufrufen oder eine Website im Browser öffnen usw.

Beispiel:

```
Intent intent = new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.scyte.eu"));
startActivity(intent);
```

Mit diesem impliziten Intent starten Sie den Browser und rufen die Web-Adresse www.scyte.eu auf.

Intent zum Starten der Activity hinzufügen

Durch Aufruf eines expliziten Intents starten Sie eine Activity. Fügen Sie in der switch/case-Anweisung für jede Activity-Klasse ein Intent ein.

Listing 4.4 case-Block der Text-Activity

```
case R.id.mainBottomActionBar_imageButton_textNote:
    startActivity(new Intent(this, NoteTextActivity.class));
    break;
```

■ 4.5 Activities im Manifest registrieren

Zum Abschluss müssen Sie die Activities noch im Manifest bekannt machen. Sollten Sie dies einmal vergessen, wird die Anwendung beim Aufruf der Activity einfach beendet. Weiterhin sollten Sie auch den Versions-Code und den Versionsnamen pflegen. Der Versions-Code ist immer eine ganze Zahl, und der Versionsname ist ein frei wählbarer Text.

Listing 4.5 Android Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="eu.scyte.notes"
    android:versionCode="1"
    android:versionName="Version 0.1" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="eu.scyte.notes.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="NoteTextActivity"></activity>
        <activity
            android:name="NoteSketchActivity"></activity><activity
            android:name="NoteVoiceActivity"></activity>

    </application>
</manifest>
```

5

Menüs und Ressourcen

In Kapitel 4 haben Sie die Grundoberflächen der App erstellt. Diese stellen jedoch noch nicht das endgültige Design der App-Oberfläche dar. Deshalb werden wir in diesem Kapitel das Layout der App weiter anpassen.



Die Beispieldateien zu diesem Kapitel finden Sie unter
www.downloads.hanser.de im Unterordner *scytenotes 0.2*.

■ 5.1 Menüs

Die Menüs haben sich im Laufe der einzelnen Android-Versionen am stärksten geändert. Seit der Ice Cream Sandwich-Version sind auf den Geräten die Hardwarebuttons der Menüs verschwunden. Ebenso wurde die Darstellung durch die Einführung der ActionBar stark verändert. Es gibt nun zwei Arten von Menüs, das Optionsmenü und das Kontextmenü. Das Optionsmenü ist mit einer klassischen Menüleiste anderer Anwendungen vergleichbar und sollte nur Funktionen enthalten, die für die aktuelle Activity sinnvoll sind. Zusätzlich ist eine Hilfe- und Informationsseite der Anwendung sinnvoll. Das Kontextmenü ist in klassischen Anwendungen meist durch einen Rechtsklick auf ein Objekt aufrufbar. Dieser Rechtsklick wird in Android durch das lange Antippen eines Objekts auf der Oberfläche ersetzt.

Im folgenden Abschnitt erläutern wir die Erstellung des Optionsmenüs.

5.1.1 Optionsmenü erstellen

Die Erstellung eines Menüeintrages in der ActionBar ist schnell erledigt. Wenn Sie die Version 1 der Anwendung ein wenig getestet haben, werden Sie feststellen, dass der Projektassistent schon einen Menüeintrag in der Main-Activity erstellt hat. Diesen werden wir nun ändern und weitere Einträge hinzufügen. Wir wollen die Menüeinträge `add` (= Hinzufügen einer neuen Text-Notiz), `help` (= Aufruf der Hilfeseite) und `about` (= Aufruf der Infoseite) anlegen.

1. Öffnen Sie durch Doppelklick die Datei *main.xml* im Ordner */res/values*.
2. Das Main-Menü enthält schon einen Eintrag, der automatisch durch den Assistenten bei der Projektanlage erstellt wurde. Löschen Sie diesen Eintrag.
3. Jetzt legen Sie einen neuen Eintrag an, in dem Sie auf **Add...** klicken und „*Create a new Element at the top level, in Menu*“ und *Item* wählen. Editieren Sie die Attribute, wie in Bild 5.1 zu sehen.

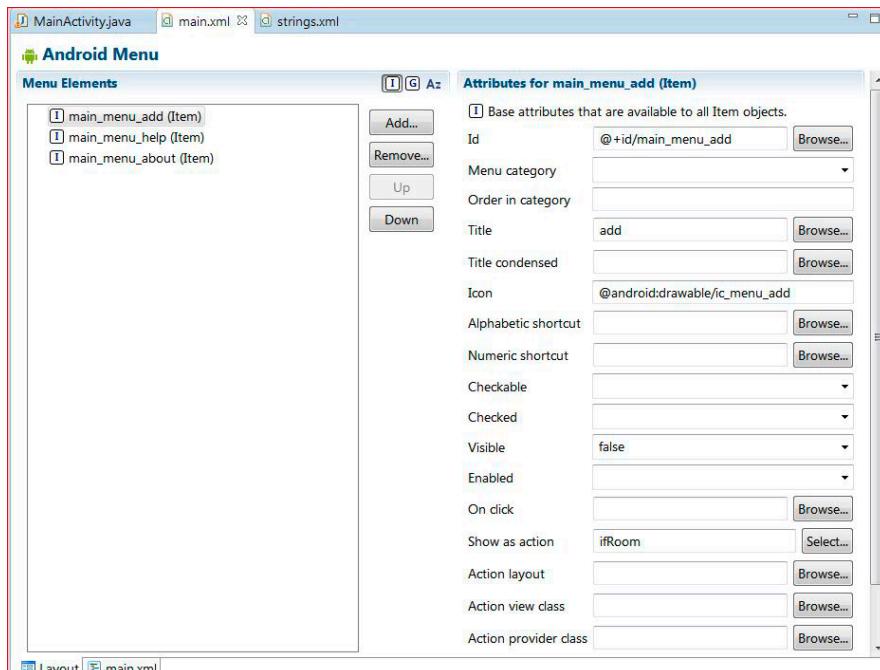


Bild 5.1 Menü erstellen

4. Wiederholen Sie den Vorgang für die Menüeinträge *help* und *about*.

Listing 5.1 Menü main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/main_menu_add"
        android:showAsAction="ifRoom"
        android:title="add"
        android:icon="@android:drawable/ic_menu_add"/>
    <item android:id="@+id/main_menu_help"
        android:icon="@android:drawable/ic_menu_help"
        android:title="help"
        android:showAsAction="never"/>
    <item
        android:id="@+id/main_menu_about"
        android:title="about"
        android:showAsAction="never"
        android:icon="@android:drawable/ic_dialog_info" />
</menu>
```

Das Attribut `android:showAsAction=" "` legt fest, wann und wie der jeweilige Menüeintrag in der ActionBar angezeigt wird.

- `never`: Der Menüeintrag wird nie in der ActionBar angezeigt, nur durch Aufruf des Optionsmenüs.
- `ifRoom`: Der Menüeintrag wird in der ActionBar angezeigt, sofern genügend Platz vorhanden ist. Ist dort nicht genügend Platz vorhanden, wird der Eintrag in der Liste des Optionsmenüs angezeigt.
- `always`: Der Eintrag wird immer in der ActionBar angezeigt.
- `withText`: Es wird nur der Text angezeigt, nicht das zugewiesene Icon.
- `collapseActionView`: Dieser Eintrag ist für die Darstellung von View-Elementen zuständig, wie z.B. ein EditText-Element zur Eingabe von Suchwörtern. In der ActionBar wird nur der Text oder das Icon angezeigt und bei Betätigen dieser Schaltfläche erscheint das Eingabe-Element.

Klick auf Menü-Item im Code behandeln

Die Ereignisbehandlung der Menüeinträge in der ActionBar erfolgt ähnlich wie das Klick-Ereignis der Buttons. Seit Einführung der ActionBar spricht man allerdings nicht mehr von Events, sondern von Actions. Um die Actions behandeln zu können, werden die Methoden `onCreateOptionsMenu(Menu menu)` und `onOptionsItemSelected(MenuItem item)` benötigt. Die Methode `onCreateOptionsMenu(Menu menu)` ist für die Erstellung des Menüs zuständig. Die Methode `onOptionsItemSelected(MenuItem item)` wird immer dann ausgeführt, wenn ein Menü-Item betätigt wurde. In der Methode erfolgt die Ermittlung, welcher Menüeintrag betätigt wurde. In der Verwendung ist sie analog zur schon bekannten Methode `onClick(View v)`.

1. Fügen Sie die Methode `onOptionsItemSelected(MenuItem item)` im Code der *Main Activity* unterhalb der `onCreateOptionsMenu`-Methode hinzu. Geben Sie dazu einfach `onOptionsItemSelected` ein, drücken dann die Tastenkombination STRG+LEER und wählen die Methode `onOptionsItemSelected(MenuItem item)` aus. Alternativ können Sie auch im Code das Kontextmenü (Rechtsklick) aufrufen und *Override/Implement Methods* benutzen.
2. Erstellen Sie wieder eine switch/case-Anweisung, um auszuwerten, welcher Menüeintrag gewählt wurde. Auch hier erfolgt die Unterscheidung anhand der *Id* des Items.
3. Nun verschieben Sie den Aufruf `startActivity(new Intent(this, NoteTextActivity.class))`; aus der Methode `onClick(View v)` in den *case-Zweig* des Optionsmenüs, der das *add-Item* auswertet. Beim Betätigen des Eintrags *add* im Optionsmenü wird jetzt die NoteTextActivity gestartet.

Listing 5.2 Methode `onOptionsItemSelected()` *MainActivity*

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.main_menu_add:  
            startActivity(new Intent(this, NoteTextActivity.class));  
            return true;
```

```

        case R.id.mainBottomActionBar_imageButton_scetchNote:
            return true;
        case R.id.mainBottomActionBar_imageButton_voiceNote:
            return true;
    }
    return super.onOptionsItemSelected(item);
}

```

■ 5.2 String-Ressourcen

In Kapitel 4 und in Abschnitt 5.1 haben Sie Steuerelementen einen Text zugewiesen, die dadurch gelb gekennzeichnet wurden. Bei der Erstellung von Oberflächen werden ausschließlich String-Ressourcen genutzt und so genannte *hardcoded-Strings* vermieden. Das Konzept, das dahintersteht, ist recht einfach zu verstehen. Alle verwendeten Zeichenketten, die zur Anzeige gelangen, werden in einer eigenständigen xml-Datei als Key-Value-Paar festgelegt, und es wird nur der Key referenziert. Dadurch ist es sehr einfach möglich, die Zeichenketten schnell und effizient zentral zu verwalten, ohne den gesamten Code oder die Layouts zu durchsuchen. Außerdem können diese Zeichenketten an vielen Stellen verwendet werden. String-Ressourcen werden in der Datei `strings.xml` im Ordner `/res/values` verwaltet. In dieser Datei können auch String-Arrays vorkommen, die z. B. bei Listen genutzt werden (siehe Abschnitt 6.3). Auch diese können wieder unterschiedliche Kennzeichner besitzen, z. B. für eine andere Sprache.

5.2.1 Eine String-Ressource anlegen und referenzieren

1. Öffnen Sie die Datei `strings.xml` im Ordner `/res/values`.
2. Erstellen Sie die neuen Einträge für *add*, *help* und *about*. Der Key ist bei diesen auch gleichzeitig der Value. Klicken Sie auf **ADD...**, wählen **STRING** und dann **OK**. Jetzt vergeben Sie den Namen (Key) für diesen String, geben den Text ein und nehmen eine Speicherung vor.
3. Da Sie schon einmal dabei sind, legen Sie die Texteinträge für die `TextView`-Elemente der unteren Navigationsleiste gleich mit an.
4. Löschen Sie alle nicht benutzten Einträge.
5. Weisen Sie jetzt den Menü-Items und den `TextViews` die entsprechenden String-Ressourcen zu. Wählen Sie einen Menüeintrag in `/res/menu/main.xml` und weisen dem Attribut `Title` die String-Ressource zu.

Listing 5.3 String-Ressourcen `strings.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<resources>

```

```
<string name="app_name">scyte notes</string>
<string name="add">add</string>
<string name="about">about</string>
<string name="help">help</string>
<string name="write">Write</string>
<string name="scetch_highlight">Scetch & Highlight</string>
<string name="record">Record</string>

</resources>
```



HINWEIS: Sonderzeichen oder Umlaute müssen HTML- bzw. Unicode-kodiert sein. Sie sollten bei der Erstellung den grafischen Editor nutzen, der die Kodierung für Sie übernimmt.

■ 5.3 Drawable-Ressourcen

Eine Drawable-Ressource ist nicht nur eine Bild-Datei des Typs PNG oder JPG. Drawable-Ressourcen werden auch durch XML-Dateien beschrieben. Durch diesen Mechanismus lassen sich einzelne View-Elemente, wie Rahmen oder Zustände, darstellen. Eine vollständige Übersicht der verfügbaren Drawables finden Sie unter <http://developer.android.com/guide/topics/resources/drawable-resource.html>.

In diesem Abschnitt wollen wir zeigen, wie Sie mit Drawables das untere Navigationsmenü ein wenig schöner darstellen.



Die in unserer Beispielanwendung verwendeten Bilder und Icons sind frei verfügbar und stammen aus zwei Quellen.

- Quelle 1: www.iconpharm.com
Falls Sie dieses Icon verwenden wollen, müssen Sie in Ihrer Anwendung darauf hinweisen.
- Quelle 2: <http://developer.android.com/design/downloads/index.html>
Wir haben hier das ActionBar-Icon-Pack verwendet.

Weitere Infos und Icons finden Sie unter <http://icons8.com>.

5.3.1 Drawable-Ressourcen verwenden

Bisher verwenden die ImageButtons in der unteren Navigationsleiste als Image-Ressource nur Standard-Ressourcen. Selbst wenn Standard-Ressourcen von Bildern verwendet werden, wird empfohlen, jedes verwendete Image eigenständig mit seiner Anwendung auszuliefern, denn diese Standard-Ressourcen können von der Version abhängen und in anderen nicht verfügbar sein. Der Standardordner für Drawables ist `/res/drawable`. Hier werden

alle Drawables abgelegt, die keine Ausrichtung, Sprache, Auflösung etc. spezifizieren. Im Projekt finden Sie Drawable-Ordner, die zusätzliche Erweiterungen besitzen, wie z.B. *drawable-hdpi*. In diesen Ordner werden Ressourcen abgelegt, die für die jeweilige Auflösung optimiert sind. Sie können auch andere Kennzeichner verwenden, z.B. die Ausrichtung Landscape. In diesem Ordner */res/drawable-land* werden dann alle Drawables für den *Landscape Mode* abgelegt. Eclipse und das ADT-Plug-in machen es einem sehr einfach, neue Ressourcen, auch mit Kennzeichnern, anzulegen.



Die von uns verwendeten Bilder finden Sie unter www.downloads.hanser.de im Projektordner zu diesem Kapitel.

5.3.2 ActionBar-Icons erstellen und verwenden

Im DesignGuide gibt es einen eigenen Abschnitt für Größen und Farben von Icons (siehe <http://developer.android.com/design/style/iconography.html>).

Um das passende ActionBar-Icon zu erstellen, muss man kein Grafiker sein. Auch hier bietet Google mit dem Android Asset Studio Unterstützung an. Das Asset Studio ist eine Sammlung von Tools, die das Erstellen von Themes und Icons für Android stark vereinfachen. Diese funktionieren jedoch nur zuverlässig im hauseigenen Browser Chrome. Das Asset Studio können Sie unter <http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html> erreichen. Am Beispiel des Image-Buttons für die Sprach-Notiz der Navigationsleiste zeigen wir kurz die Nutzung.

1. Öffnen Sie im Browser Chrome die Adresse des Asset Studios.
2. Wählen Sie jetzt *Action bar and tab icons*.
3. Hier sagt ein Bild mehr als tausend Worte (siehe Bild 5.2).
4. Entpacken Sie die heruntergeladene Zip-Datei und kopieren die Bild-Dateien in den jeweils entsprechenden Ordner in Eclipse.
5. Weisen Sie nun den ImageButtons die neuen Ressourcen zu. Das zu ändernde Attribut des Buttons ist: `android:src=" " .` Sie sollten das Layout *activity_main.xml* geöffnet haben. Navigieren Sie zum ImageButton und weisen Sie die neue Ressource zu.

Listing 5.4 ImageButton voiceNote in activity_main.xml

```
<ImageButton
    android:id="@+id/mainBottomActionBar_imageButton_voiceNote"
    android:layout_width="40dp"
    android:layout_height="40dp"
    android:layout_gravity="center"
    android:onClick="onClick"
    android:scaleType="center"
    android:src="@drawable/ic_action_record" />
```

6. Führen Sie dies auch für die anderen beiden ImageButtons durch.

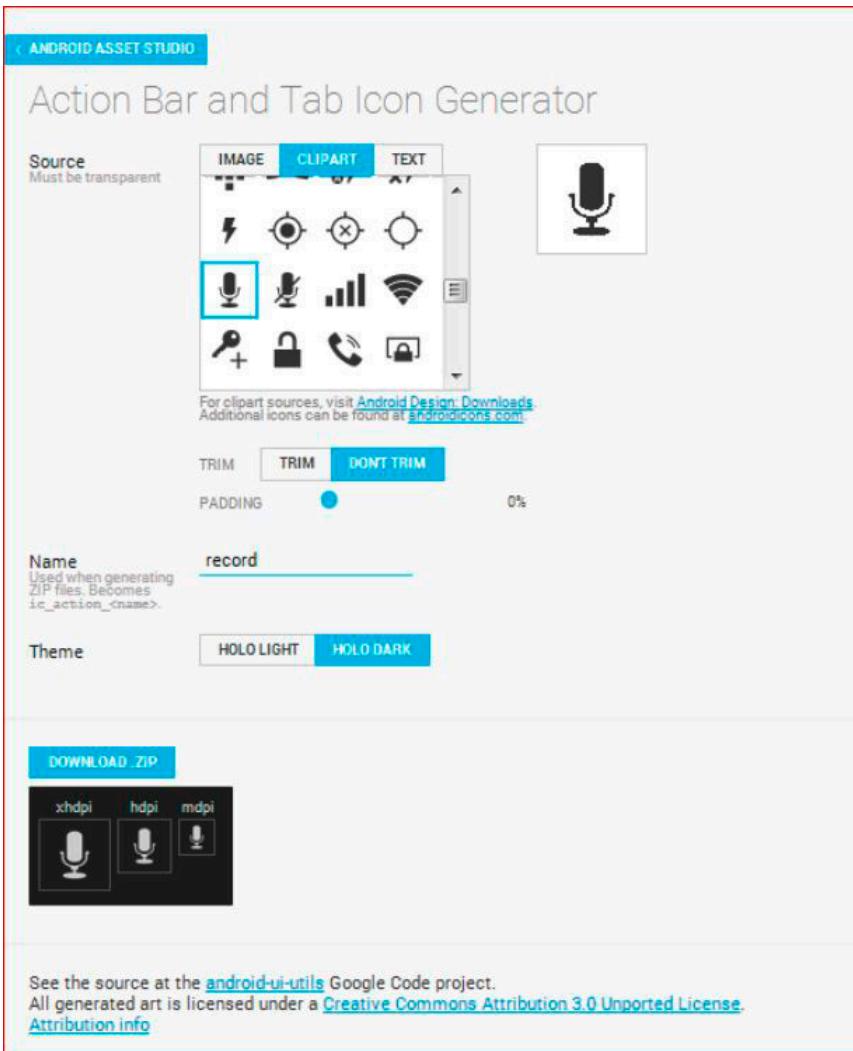


Bild 5.2 Erstellung eines Icons mit dem Android Asset Studio

5.3.3 xEffekte durch Drawables

Wir verfolgen zwei Ziele. Zum einen sollen die ImageButtons der Navigationsleiste etwas angenehmer dargestellt werden, und zum anderen soll ein blauer Rahmen um die Buttons erscheinen, wenn diese ausgewählt wurden. Es gibt leider kein Attribut *Border*, um einen Rahmen um ein View-Elemente zu zeichnen. Auch das naheliegende Attribut *Background* ist nicht praktikabel, würde dieses benutzt, verliert der Button seinen 3D-Effekt und wird nur noch flach dargestellt. Jedoch gibt es für diese Anforderung eine Lösung.

1. Zunächst legen Sie eine neue Ressource `image_button_checked.xml` im Ordner `/res/drawable` an. Achten Sie darauf, dass als Root-Element `Shape` ausgewählt ist.
2. Nun geben Sie der Datei folgenden Inhalt (siehe Listing 5.5).

Listing 5.5 Ressource `image_button_default.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <stroke
        android:width="2dp"
        android:color="#4876FF" />
    <gradient
        android:startColor="#363636"
        android:endColor="#3B3B3B"
        android:angle="90" />
    <corners
        android:topLeftRadius="7dp"
        android:topRightRadius="7dp"
        android:bottomLeftRadius="7dp"
        android:bottomRightRadius="7dp" />
    <size
        android:width="40dp"
        android:height="40dp" />
</shape>
```

3. Jetzt legen Sie eine weitere Ressource `image_button_default.xml`, ebenfalls in `/res/drawable`, an. Kopieren Sie den Inhalt der vorherigen Datei in diese hinein. Hier entfernen Sie das Element `<stroke>`.

Erläuterung: Mit der XML-Ressource (`image_button_checked.xml`) beschreiben Sie das Aussehen eines Rechtecks. Das Rechteck ist 40 dp hoch und breit, die Fläche wird mit einem Farbverlauf gefüllt, die Ecken werden mit einem Radius gezeichnet, der äußere Rahmen ist 2 dp stark und hat die Farbe Blau. Durch das Root-Element `<shape>` definieren Sie, dass diese XML-Ressource eine Form beschreibt, mit `android:shape="rectangle"` legen Sie die Form eines Rechtecks fest. Im Tag `<stroke>` wird die äußere Linie mit Größe und Farbe (#4876FF = HTML-Code der Farbe Blau) definiert. Das Element `<gradient>` definiert einen Farbverlauf durch `startColor` und `endColor`, außerdem wird mit `angle` der Verlaufswinkel und mit `<corners>` der Radius der Ecken festgelegt. Mit `<size>` legen Sie die Größe fest. Elemente, die nicht gezeichnet werden sollen, werden auch nicht definiert.

4. Im Layout der Activity (`activity_main.xml`) müssen Sie nun den ImageButtons den Background `image_button_default` zuweisen: `android:background="@drawable/image_button_default"`
5. Nun muss der Code der Main-Activity (`MainActivity.java`) so angepasst werden, dass bei einem Klick auf einen Button, dem betätigten Button der Background `image_button_checked` zugewiesen und der Background der anderen Buttons auf default gesetzt wird. Um dies zu erreichen, müssen Sie zuerst die ImageButtons bekannt machen. Dies erfolgt am besten in der `onCreate`-Methode. Wir müssen in der `onClick(View v)` auf die ImageButtons zugreifen. Darum deklarieren Sie jeden ImageButton als Member-Variable der Klasse, wie in Listing 5.5. In der `onCreate(Bundle)`-Methode referenzieren wir nun die

Member-Variablen auf die View-Elemente des Layouts, d.h. jede Variable erhält eine eindeutige Referenz auf das View-Element, die Referenz zeigt auf die *Id* der View (siehe Listing 5.7).

6. Die `onClick(View v)`-Methode wird aufgerufen, wenn ein Klick-Event erfolgt. Da liegt es nahe, in dieser Methode auch die Eigenschaften der ImageButtons zu verändern. In der switch/case-Anweisung wird dem Button, der das `onClick`-Event sendet, also der betätigt wurde, der checked-Hintergrund zugewiesen. Die anderen beiden Buttons erhalten den default-Hintergrund. Die Eigenschaft `Background` ändern Sie durch `.setBackgroundResource(...)`; (siehe Listing 5.8)

Listing 5.6 Deklaration der ImageButton-Variablen

```
public class MainActivity extends Activity {  
    ImageButton imgButtonTextNote;  
    ImageButton imgButtonScetchNote;  
    ImageButton imgButtonVoiceNote;
```

Listing 5.7 Member-Variablen auf View-Elemente referenzieren

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    imgButtonTextNote = (ImageButton)  
findViewById(R.id.mainBottomActionBar_imageButton_textNote);  
    imgButtonScetchNote = (ImageButton)  
findViewById(R.id.mainBottomActionBar_imageButton_scetchNote);  
    imgButtonVoiceNote = (ImageButton)  
findViewById(R.id.mainBottomActionBar_imageButton_voiceNote);  
}
```

Listing 5.8 Hintergrund der ImageButtons per Code ändern

```
public void onClick(View v) {  
    switch (v.getId()) {  
        case R.id.mainBottomActionBar_imageButton_textNote:  
            imgButtonTextNote.setBackgroundResource(R.drawable.image_button_checked);  
            imgButtonScetchNote.setBackgroundResource(R.drawable.image_button_default);  
            imgButtonVoiceNote.setBackgroundResource(R.drawable.image_button_default);  
            break;
```



HINWEIS: Farbwerte werden in Android durch HTML-Farbcode, einer Resourcen-Id oder den Integer-Farbwert festgelegt. Sie können Farben in einer eigenen Farb-Ressource verwalten.



HINWEIS: Denken Sie daran, den Versionscode und den Versionsnamen im Android-Manifest zu ändern (Versionscode = 1, Versionsname = Version 0.2).

6

Eigene Klassen, Listen und Adapter

In diesem Kapitel wollen wir zeigen, wie Sie auf das Dateisystem zugreifen, Daten in der Liste darstellen und mit diesen Daten arbeiten können. Weiterhin erläutern wir, wie Sie die Listeneinträge in einem eigenen Design darstellen.

■ 6.1 Eigene Klassen erstellen

Im Java-Crash-Kurs (Kapitel 3) erläuterten wir, dass Java-Klassen Baupläne für Objekte sind. Eigentlich wissen Sie bereits, wie Sie neue Klassen erstellen. Schließlich haben Sie schon die Activities erstellt und kennen Java-Klassen. Die Notizen der Beispiel-App sind ebenfalls solche Objekte. Um die Notizen ordentlich verwalten zu können, benötigen wir eine Klasse, welche die Eigenschaften und Fähigkeiten unserer Notizen festlegt. Wir erstellen in diesem Abschnitt die Klasse *Note* und werden im weiteren Verlauf des Projekts diese Klasse immer weiter anpassen. Die Notizen sind im Wesentlichen einfache Dateien. Daraus ergeben sich drei grundlegende Eigenschaften: Der Dateiname, der vollständige Dateipfad und das Dateidatum. Diese Eigenschaften werden ganz oder in abgewandelter Form genutzt. Später kommen noch einige Eigenschaften hinzu.

1. Erstellen Sie die Klasse *Note* im Ordner */src*, markieren Sie das Paket, und öffnen das Kontextmenü (Rechtsklick). Wählen Sie **New** und anschließend **Class**. Im folgenden Fenster vergeben Sie als Namen *Note*, belassen den Modifizierer auf **Public** und behalten die Superklasse **java.lang.Object** bei. Schließen Sie den Vorgang mit **Finish** ab.
2. Definieren Sie nun die Eigenschaften der Klasse *Note*. Fügen Sie im Code der *Note*-Klasse folgende drei Eigenschaften hinzu:
 - **private String mSubject**
 - **private String mFilePath**
 - **private Long mFileDate**
3. Die Eigenschaften wurden in Eclipse direkt gelb gekennzeichnet. Setzen Sie den Mauszeiger auf einen gelb markierten Bezeichner. Nun sollte Eclipse Quick Fix ein Pop-up-Fenster mit verschiedenen Optionen anzeigen (Tastenkombination STRG + 1). Wählen Sie den Eintrag *Create Getter and Setter for ...*. Sollte bei Ihnen, aus welchen Gründen auch immer, Quick-Fix nicht funktionieren, können Sie natürlich die Getter- und Setter-Methoden

trotzdem von Eclipse generieren lassen, und zwar mit der Tastenkombination ALT + SHIFT + S und dann GENERATE GETTERS AND SETTERS.

Listing 6.1 Klasse Note

```
package eu.scyte.notes;
public class Note {
    private String mSubject;
    private String mFilePath;
    private Long mFileDate;

    public String getSubject() {
        return mSubject;
    }
    public void setSubject(String subject) {
        mSubject = subject;
    }
    public String getFilePath() {
        return mFilePath;
    }
    public void setFilePath(String filePath) {
        mFilePath = filePath;
    }
    public Long getFileDate() {
        return mFileDate;
    }
    public void setFileDate(Long fileDate) {
        mFileDate = fileDate;
    }
}
```

Erläuterung

Sie haben nun den Bauplan der Notizen erstellt. Jedes Mal, wenn eine neue Notiz, ein Note-Objekt, erstellt wird, besitzt dieses Objekt die hier festgelegten Eigenschaften.

Die Klasse *Note* verfügt jetzt über die drei Eigenschaften *mSubject* des Typs *String*, *mFilePath* ebenfalls des Typs *String* und *mFileDate* des Typs *long*. Die Attribute (Eigenschaften) wurden nicht willkürlich gewählt. Zielsetzung war, eine Klasse *Notizen* zu erstellen, die alle drei Arten unserer Notizen (Text, Bild und Audio) erzeugen kann.

Die Attribute *mSubject* und *mFileDate* sind lediglich vorhanden, damit diese Informationen einfach zur Anzeige gebracht werden können und nicht jedes Mal ein Zugriff auf das Dateisystem erfolgen muss. Alle Notizen werden in Dateien gespeichert, und die hier erzeugten Attribute lassen sich aus den Dateieigenschaften ableiten. Der Betreff (*mSubject*) ist der Dateiname ohne Dateiendung, das Attribut *mFileDate* ist das Änderungsdatum der Datei und *mFilePath* ist der absolute Pfad im Dateisystem der Datei. Jede Datei, ausgehend vom Datei-Pfad, kann somit nur einmal im Dateisystem vorkommen. Der Datei-Pfad ist ein eindeutiger Identifizierer, über den später die Zuordnung zu den Erinnerungen in der Datenbank erfolgt.

Sie wundern sich wahrscheinlich, warum wir die Eigenschaften *mSubject* und nicht einfach *subject* genannt haben. Dies hat natürlich einen Grund. In Java ist es gängige Praxis, Variablen einer Klasse, so genannte *Member-Variablen*, ein *m* voranzustellen, um diese von Parametern einer Methode unterscheiden zu können.

Sie haben den Attributen `mSubject`, `mFilePath` und `mFileDate` den Zugriffsmodifikator *private* vorangestellt. Dieser bewirkt, dass der Zugriff auf die Attribute nur innerhalb der Klasse möglich ist. Dies ist auch der Grund, warum Sie die Getter- und Setter-Methoden in der Klasse erzeugen mussten. Nur durch diese Methoden ist es nun möglich, auf die Eigenschaften der Klasse zuzugreifen und diese zu verändern. Getter- und Setter-Methoden werden häufig eingesetzt, um auf die geschützten Eigenschaften zugreifen zu können. Die Sichtbarkeit der Methoden ist davon abhängig, woher der Zugriff erfolgt. Oft werden in den Getter- und Setter-Methoden Validierungen durchgeführt.

Tabelle 6.1 Zugriffsrechte in Java

private	Klasse			
default (kein Schlüsselwort)	Klasse	Paket		
protected	Klasse	Paket	Unterklassen	
public	Klasse	Paket	Unterklassen	alle



HINWEIS: Das `default`-Zugriffsrecht besitzt kein eigenes Schlüsselwort. Dieses wird auch als „package private“ bezeichnet, da ein Zugriff innerhalb des Paketes erfolgen kann. Die Deklaration erfolgt einfach durch Typ und Namen, z.B. `String` betreff;. Außerdem ist zu beachten, dass Unterpakete hier als andere Pakete gelten, und ein Zugriff von Unterpaketen nicht gestattet ist. Dies gilt auch für das Zugriffsrecht `protected`.

■ 6.2 Enumerationen

Um die Notiz-Typen einfacher unterscheiden zu können, verwenden wir Enumerationen (Aufzählungen). Es ist durchaus einzuwenden, dass diese Unterscheidung anhand der Dateiendung realisierbar wäre, der Vollständigkeit halber haben wir jedoch diese Art der Unterscheidung gewählt. Wir verwenden die Enumeration, um den Notiz-Typ und den Typen der Erinnerung festzulegen bzw. zu ermitteln. Der Enum `NoteType` ist sehr einfach zu erkennen. Es existieren drei Typen von Notizen, und diese zählen wir im Enum auf. Bei den Erinnerungen gehen wir von der Notiz aus und überlegen, welche Erinnerungs-Typen die Notiz haben kann. Eine zeitbasierte Erinnerung, also *time*, eine ortsbasierte Erinnerung, also *location* oder gar keine Erinnerung, also *none*. Die Erstellung ist einfach.

1. Markieren Sie im Package-Explorer das Package und rufen mit Rechtsklick das Kontextmenü auf, wählen Sie `New` und dann `Enum`.
2. Jetzt vergeben Sie den Namen `NoteType` und schließen den Vorgang mit `Finish` ab.
3. In der Code-Ansicht geben Sie die Typen *none*, *time*, *location* ein und speichern die Java-Datei.

4. Den Vorgang wiederholen Sie für den Erinnerungs-Typen. Als Namen vergeben Sie `ReminderType`, die Typen sind *none*, *time*, *location*. Nun speichern Sie die Datei ebenfalls.

Listing 6.2 Enum NoteType

```
package eu.scyte.notes;

public enum NoteType {
    text, sketch, voice
}
```

Listing 6.3 Enum ReminderType

```
package eu.scyte.notes;

public enum ReminderType {
    none, location, time
}
```



HINWEIS: Achten Sie bei der Erstellung darauf, dass Sie die Java-Dateien im richtigen Paket erstellen: `eu.scyte.notes`.

Sie haben jetzt zwar die Notiz- und Erinnerungs-Typen erstellt, allerdings können Sie bei einem Notiz-Objekt nicht feststellen, was für ein Notiz-Typ es ist, und was für einen Erinnerungs-Typen es hat. Daher müssen Sie die Klasse `Note` um die Eigenschaften `NoteType` und `ReminderType` erweitern. Die Getter- und Setter-Methoden sollten Sie gleich mit anlegen.

Listing 6.4 Erweiterung der Klasse Note mit NoteType und ReminderType

```
private NoteType mNoteType;
private ReminderType mReminderType;

public NoteType getNoteType() {
    return mNoteType;
}
public void setNoteType(NoteType noteType) {
    mNoteType = noteType;
}
public ReminderType getReminderType() {
    return mReminderType;
}
public void setReminderType(ReminderType reminderType) {
    mReminderType = reminderType;
}
```

■ 6.3 Arrays, Listen und Adapter

Überall in Android kommen Listen zum Einsatz, egal, ob Sie das Telefonbuch, die E-Mail-App oder die Einstellungen öffnen. Der Umgang mit Listen gehört in Android zum täglichen Brot. Wie schon bei den Activities aufgezählt, gibt es sogar Activities die für die Darstellung

von Listen optimiert sind. In diesem Abschnitt wollen wir die Grundlagen für den Umgang mit Listen erklären.

Listen: Listen kann man mit Arrays vergleichen. Eine Liste listet eine beliebige Menge von Objekten beliebigen Typs auf. Auf diese Elemente der Liste kann über einen Index oder sequenziell zugegriffen werden. Listen unterscheiden sich jedoch dahingehend von Arrays, dass bei der Initialisierung einer Liste die Größe nicht wie bei einem Array angegeben werden muss. Listen verbrauchen mehr Speicher, und der Zugriff auf die Elemente einer Liste ist aufwändiger. Der einfachste Typ einer Liste ist eine *ArrayList*. Je nach Verwendung werden Arrays oder Listen eingesetzt.

Adapter: Ein Adapter oder besser *ListAdapter* befüllt die Liste mit Objekten. Auch ein einfacher String ist ein Objekt und bringt diese in der *ListView* zur Anzeige. Außerdem stellt er sicher, dass die richtigen Werte dargestellt werden. Android bringt eine Reihe von fertigen Klassen (Adaptoren) mit, die Daten wie ein String-Array (*ArrayAdapter<String>*) oder Tabellenzeilen (*SimpleCursorAdapter*) einer Datenbank anzeigen können.

6.3.1 Einfache Strings in Spinner und Liste anzeigen

Um einen schnellen Einstieg in die Grundlagen zu bekommen, machen wir einen kleinen Ausflug und erstellen zunächst ein kleines Übungs-Projekt zum Testen. Wir haben es *Training Listen* genannt. Die Erstellung eines neuen Projektes sollte ja jetzt kein Problem mehr für Sie sein.

In dieser kleinen Übung werden zwei unterschiedliche View-Elemente zur Anzeige von Listen genutzt, der *Spinner* und die *ListView*. Diese beiden bringen jeweils ein eigenes String-Array zur Anzeige. Das erste String-Array enthält Rosennamen und soll in einem Spinner dargestellt werden. Das zweite String-Array enthält alle europäischen Länder und soll unter dem Spinner in einer *ListView* angezeigt werden. Bild 6.1 zeigt das fertige Projekt *Training Listen*.

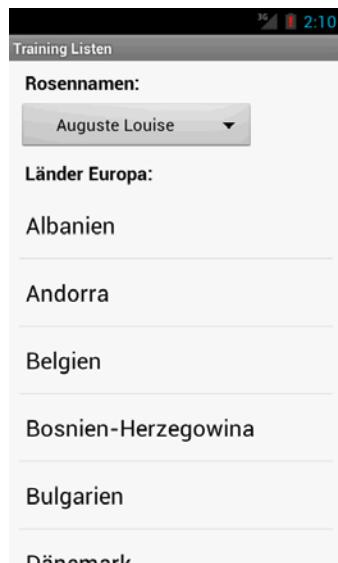


Bild 6.1
Screen Training Listen

Nun, da Sie haben das Trainingsprojekt erstellt haben, sind es nur noch ein paar Schritte.

1. Als Erstes erstellen Sie die beiden String-Arrays. String-Arrays werden in der Ressource `string.xml` im Ressourcen-Ordner `/res/values` abgelegt. Öffnen Sie die Datei, oder legen Sie diese an, falls sie nicht vorhanden ist. In Listing 6.5 sind nicht alle Items aufgeführt.

Listing 6.5 String-Arrays in strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Training Listen</string>
    <string name="spinner_label">Rosennamen:</string>
    <string name="listview_label">Länder Europa:</string>
    <string-array name="roses">
        <item >Auguste Louise</item>
        <item >Chandoz Beauty</item>
        <item >Gertrude Jekyll</item>
        <item >....</item>
    </string-array>
    <string-array name="countries_europe">
        <item >Albanien</item>
        <item >Andorra</item>
        <item >Belgien</item>
        <item >....</item>
    </string-array>
    <string name="no_data">keine Daten vorhanden</string>
</resources>
```

2. Jetzt legen Sie das Layout fest. Öffnen Sie die Layout-Datei `activity_training_listen.xml` im Ressourcen-Ordner `layout`. Zunächst ändern Sie den Layout-Typen der Activity von `RelativeLayout` in `LinearLayout`. Markieren Sie in der `Outline-View` den obersten Container `RelativeLayout`, rufen Sie das Kontextmenü (Rechtsklick) auf und wählen Sie `CHANGE LAYOUT` aus. Wählen Sie nun das `LinearLayout` (vertikal). Fügen Sie eine `TextView` als Label für den Spinner, den Spinner, wieder eine `TextView`, die `ListView` und zum Abschluss eine `TextView` hinzu. Vergeben Sie die `Id android:id="@+id/android:list"` an die `ListView` und die `Id android:id="@+id/android:empty"` an die unterste `TextView`. Die `Id` sollten Sie in der XML-Ansicht ändern, in der grafischen Oberfläche führt dies zu Fehlern.
3. Dem Attribut `android:entries=` weisen Sie den String-Array der Rosennamen zu: `android:entries="@array/roses"`. Damit haben Sie den Spinner schon fertig gestellt und bekommen die Rosennamen zur Anzeige.
4. Um die Länder Europas in der `ListView` anzuzeigen, müssen Sie den Code der Activity-Klasse anpassen. Öffnen Sie die `TrainingListen.java` im Ordner `Sourcen, /src`. Diesmal soll eine `ListActivity` verwendet werden. Ändern Sie dazu die Deklaration der Klasse `TrainingListen`, sodass diese `ListActivity` erweitert wird.

Listing 6.6 Code der Activity `TrainingListen`

```
package eu.scyte.traininglisten;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
```

```
public class TrainingListen extends ListActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_training_listen);  
  
        ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(this,  
            android.R.layout.simple_list_item_1, getResources().getStringArray(R.array.  
countries_europe));  
        setListAdapter(arrayAdapter);  
    }  
  
}
```

Erläuterung

Die `onCreate()`-Methode und die Zuweisung des Layouts durch `setContentView()` ist Ihnen bereits bekannt.



Das Trainingsprojekt können Sie einzeln downloaden. Das Projekt steht unter www.downloads.hanser.de im Unterordner `kap3_TrainingListen` zur Verfügung.

6.3.2 Eigene Objekte in einer ListView anzeigen

Damit Sie Daten in einer `ListView` darstellen können, wird eine passende Datenquelle benötigt. Datenquellen für Listen sind Arrays, Listen (List) oder auch Datenbanktabellen. Zunächst wollen wir eine Liste von Notizen in der Übersicht anzeigen. Um Notizen, also Note-Objekte, anzeigen zu können, müssen wir eine Liste mit Notizen erstellen. Diese Liste wird dann mithilfe eines Adapters an die `ListView` gebunden.

Wir erweitern die `onCreate`-Methode der `Main-Activity`-Klasse auf folgende Weise:

1. Als Erstes referenzieren wir die `ListView` in der Activity `ListView noteListView = (ListView) findViewById(R.id.main_listView);`. Sie werden sich sicherlich fragen, warum immer `(ListView)` der Methode `findViewById()` vorangestellt ist. Das View-Element, das durch `findViewById()` referenziert wird, muss auf den Typen der View-Variablen umgewandelt (*gecastet*) werden. Diese Typumwandlung ist verpflichtend bei der Referenzierung mit `findViewById()`.
2. Nun erstellen wir eine Liste des Typs Notiz, erzeugen die Notizen und fügen sie der Liste hinzu. Die Liste wird durch `List<Note> listNotes = new ArrayList<Note>();` erzeugt.
3. Jetzt wird ein Listadapter erzeugt und initialisiert. Der Adapter erhält folgende Parameter: Den Kontext, das Layout, welches die Note-Objekte anzeigt, und die Liste, die angezeigt werden soll: `ListAdapter listAdapter = new ArrayAdapter<Note>(this, android.R.layout.simple_list_item_1, listNotes);`.
 - *Context*: Ist der aktuelle Kontext der Activity, deshalb `this`.
 - *Layout*: Ist ein vorgefertigtes ListItem von Android, `android.R.layout.simple_list_item_1`.
 - *Liste*: Ist die Liste mit den Notiz-Objekten, `listNotes`.

4. Zum Abschluss wird die *ListView* an den Adapter durch `noteListView.setAdapter(listAdapter);` gebunden.

Listing 6.7 `onCreate`-Methode der Klasse `MainActivity`

```

@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imgButtonTextNote = (ImageButton)
findViewById(R.id.mainBottomActionBar_imageButton_textNote);
        imgButtonScetchNote = (ImageButton)
findViewById(R.id.mainBottomActionBar_imageButton_scetchNote);
        imgButtonVoiceNote = (ImageButton)
findViewById(R.id.mainBottomActionBar_imageButton_voiceNote);
        noteListView = (ListView) findViewById(R.id.main_listView);
        listNotes = new ArrayList<Note>();
        Note note = new Note();
        note.setSubject("Note 1");
        listNotes.add(note);
        note = new Note();
        note.setSubject("Note 2");
        listNotes.add(note);
        note = new Note();
        note.setSubject("Note 3");
        listNotes.add(note);
        note = new Note();
        note.setSubject("Note 4");
        listNotes.add(note);
        note = new Note();
        note.setSubject("Note 5");
        listNotes.add(note);
        ListAdapter listAdapter = new ArrayAdapter<Note>(this,
            android.R.layout.simple_list_item_1, listNotes);
        noteListView.setAdapter(listAdapter);
    }
}

```

Was geschieht in der `onCreateMethod()`? (Die Zuweisung und Referenzierung von Layout-Ressourcen werden hier nicht mehr erläutert.)

Mit `listNotes = new ArrayList<Note>();` erstellen Sie eine Liste, die die Notiz-Objekte später auflisten wird. Die Liste ist eine typisierte `ArrayList`, die Objekte des Typs Notiz aufnehmen kann. Im nächsten Schritt werden fünf Notiz-Objekte erstellt, deren Eigenschaft `subject` mit `note.setSubject()` einen Wert zugewiesen wird, und mit `listNotes.add()` wird das Notiz-Objekt in die Liste eingefügt. Jetzt enthält die Liste `listNotes` fünf Notiz-Objekte, auf die z.B. über den nullbasierten Index der Liste zugegriffen werden kann.

Nun wird der Listadapter mit `ListAdapter listAdapter = new ArrayAdapter...` erstellt. Da unsere Notiz-Objekte sich in einer `ArrayList` des Typs `Note` befinden, wird ein `ArrayAdapter` benutzt, der den Typ der Objekte entgegennimmt.

Die `ListView` `noteListView` wird an den `ListAdapter listAdapter` gebunden. Der ListAdapter wird in diesem Fall durch drei Parameter initialisiert. Diese Parameter sind:

- `this`: Der Kontext, in dem der Adapter genutzt wird, hier die Activity
- `android.R.layout.simple_list_item_1`: Das Layout-Element, das ein einzelnes `ListItem` anzeigt. Dieses Layout enthält nur eine einfache `TextView`.
- `listNotes`: Die Liste mit den Objekten, die angezeigt werden sollen

Als letzten Schritt wird die ListView durch `noteListView.setAdapter(listAdapter);` an den Adapter gebunden. Die `setAdapter`-Methode erwartet immer den Adapter als Parameter.

Wenn Sie das Projekt zum jetzigen Zeitpunkt ausführen, erhalten Sie eine Ansicht wie in Bild 6.2 zu sehen. Da unsere Liste nun nicht mehr nur einfache String-Objekte enthält, sondern komplexe Objekte, zeigt das im Adapter festgelegte Layout die interne *Id* der Notiz-Objekte an.

Was sehen Sie also? Sie sehen in der Liste jedes einzelne Note-Objekt mit seiner *Id*. Die Notizen-Objekte besitzen mehrere Attribute, die in der Notizen-Klasse festgelegt wurden. Die Anzeige ist also völlig richtig. Allerdings kann niemand etwas mit dieser Ausgabe anfangen.

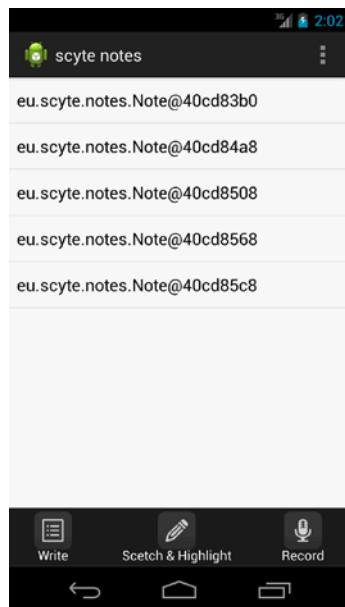


Bild 6.2
Screen Notizenübersicht

Deswegen werden wir im folgenden Abschnitt ein Layout für die Darstellung einer einzelnen Notiz erstellen, das die gewünschten Informationen anzeigt, und einen eigenen Adapter für das Layout.

6.3.3 ListItem und ListAdapter für eigene Objekte

Die Notizen sollen in einem eigenen Layout dargestellt werden. Der Notiz-Name, das Änderungsdatum, der Erinnerungsinhalt und der Typ der Erinnerung sollen wie in Bild 6.3 dargestellt werden. Das Erstellen des Item-Layouts sollte Ihnen nun ja keine Probleme mehr bereiten.

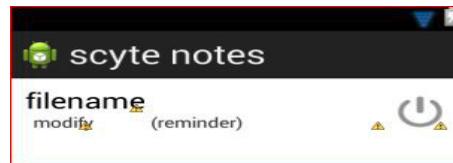


Bild 6.3 Bearbeitungsansicht ListItem der Notizen

Listing 6.8 ListItem: listitem_note.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/listItem"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <TableLayout
        android:id="@+id/listItem_Grid"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"
        android:layout_marginLeft="6dp"
        android:layout_marginRight="8dp"
        android:layout_toLeftOf="@+id/listItem_imageView_noteReminder" >

        <TextView
            android:id="@+id/listItem_textView_subject"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="-1dp"
            android:gravity="top"
            android:singleLine="true"
            android:text="filename"
            android:textAppearance="?android:attr/textAppearanceListItemSmall" />

        <TableRow
            android:id="@+id/row1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >

            <TextView
                android:id="@+id/listItem_textView_modifydate"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="8dp"
                android:hint="(modify)"
                android:singleLine="true"
                android:textAppearance="?android:attr/textAppearanceSmall" />

            <TextView
                android:id="@+id/listItem_textView_reminder"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="8dp"
                android:hint="(reminder)" />

```

```
        android:singleLine="true"
        android:textAppearance="?android:attr/textAppearanceSmall" />

    </TableRow>
</TableLayout>

<ImageView
    android:id="@+id/listItem_imageView_noteReminder"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_centerVertical="true"
    android:layout_marginBottom="4dp"
    android:layout_marginRight="4dp"
    android:layout_marginTop="4dp"
    android:padding="2dp"
    android:scaleType="center"
    android:src="@drawable/icReminderOff" />

</RelativeLayout>
```

ListItem-Klasse und eigenenListAdapter erstellen

Nachdem Sie das Layout einer einzelnen Notiz erstellt haben, sind noch einige Anpassungen nötig.

1. Zunächst erstellen Sie die Klasse *ListItemView*, die von der Klasse *RelativeLayout* erbt. Aktivieren Sie im Dialog **New Class Constructors from superclass**. Es wird nur die Methode `public NoteListItemView(Context context)` benötigt.

Listing 6.9 Klasse NoteListItemView

```
package eu.scyte.notes;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class NoteListItemView extends RelativeLayout {
    TextView subject;
    TextView modifyDate;
    TextView reminder;

    public NoteListItemView(Context context) {
        super(context);
        LayoutInflater inflater = LayoutInflater.from(context);
        View listItemView = inflater.inflate(R.layout.listitem_note, null);
        subject = (TextView) listItemView.findViewById(R.id.listItem_textView_subject);
        subject.setText("");
        modifyDate = (TextView)
listItemView.findViewById(R.id.listItem_textView_modifydate);
        reminder = (TextView) listItemView.findViewById(R.id.listItem_textView_reminder);
        addView(listItemView);
    }
}
```

```

    public void setNoteItem(Note note) {
        subject.setText(note.getSubject());
    }
}

```

2. Jetzt wird der Adapter *NoteListAdapter* als innere Klasse der *MainActivity* erstellt.

Listing 6.10 Innere Klasse NoteListAdapter der MainActivity

```

private static class NoteListAdapter extends ArrayAdapter<Note> {

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Note noteItem = getItem(position);
        NoteListItemView noteListItemView = null;
        if(convertView != null){
            noteListItemView = (NoteListItemView) convertView;
        }
        else{
            noteListItemView = new NoteListItemView(getContext());
        }
        noteListItemView.setNoteItem(noteItem);
        return noteListItemView;
    }

    public NoteListAdapter(Context context, int textViewResourceId,
    List<Note> objects) {
        super(context, textViewResourceId, objects);
    }
}

```

3. Zum Abschluss ändern Sie in der *onCreateMethode* der *MainActivity* den *ListAdapter*. Jetzt sollte die neue Klasse *NoteListAdapter* als Adapter für die ListView verwendet werden:
NoteListAdapter listAdapter = new NoteListAdapter(this, 0, listNotes);.

Erläuterung

Die Klasse *NoteListItemView* gibt uns Zugriff auf die einzelnen Layout-Ressourcen. Als Parameter muss *Context* übergeben werden. Vereinfacht ausgedrückt wird der Klasse damit mitgeteilt, in welchem Rahmen oder Umgebung Sie verwendet wird. Der *LayoutInflater* füllt das Layout mit dem referenzierten Layout des ListItems auf. Im weiteren Verlauf werden die einzelnen Layout-Elemente referenziert und der Text der TextView-Elemente wird mit der Länge 0 vorbelegt (keine Zeichen). Am Ende des Konstruktors werden die Elemente durch die *addView()*-Methode der ListItemView hinzugefügt.

Die öffentlichen Methode *setNoteItem()* gibt Zugriff auf die Layout-Elemente. Die *setNoteItem()*-Methode wird im Adapter benötigt und verlangt als Parameter eine Referenz auf ein *Note*-Objekt. Innerhalb der Methode wird dem *EditText-subject* das *Note-Subject* des übergebenen Note-Objekts zugewiesen.

Die neu erstellte Adapter-Klasse *NoteListAdapter* ist eine des Typs *Note* und erweitert den *ArrayAdapter*. Die Klasse ist **private** und **statisch**, um zu verhindern, dass von der *Main Activity*-Klasse auf Methoden zugegriffen wird. Dies würde einen erhöhten Ressourcen-Verbrauch bedeuten und auch keinen Sinn machen.

Dadurch, dass der neue Adapter von der Klasse *ArrayAdapter* erbt, müssen wir uns an der Klasse *ArrayAdapter* orientieren und legen als Parameter *Context*, *int* für die *Id* der Layout-Ressource und die *List<Note>* im Konstruktor fest. Der Integer-Parameter für die Layout-Ressource wird eigentlich nicht benötigt, da das Layout in der überschriebenen Methode *getView()* festgelegt wird. Die *getView()*-Methode ist eine des Typs *View* und liefert die *noteListItemView* zurück. Als Erstes wird das Note-Objekt mit *.getItem(position)* festgelegt und im nächsten Schritt eine leere Referenz auf die *NoteListView* erstellt. In der folgenden *if*-Bedingung wird überprüft, ob die *ConvertView* nicht **null** ist. Ist die Bedingung erfüllt, wird die *ConvertView* auf unsere *noteListItemView* umgewandelt. Ist die Bedingung nicht erfüllt, wird im *else*-Zweig ein neues *noteListItemView()*-Objekt erzeugt, das den *Context* an unsere Klasse übergibt.

Die Erstellung von Objekten ist ziemlich aufwändiger, und aus diesem Grund wurde der Mechanismus mit der *ConvertView* eingeführt, um nicht immer wieder ein neues Layout-Objekt erzeugen zu müssen. Eine *ConvertView* ist im Prinzip eine abstrakte View-Klasse, die kein eigenes Layout besitzt. Sie ist im weitesten Sinne ein Platzhalter für Layout-Objekte, allerdings ein ordentliches Objekt. Wird der Adapter aufgerufen, ist die *ConvertView* noch **null**. Wird im Adapter eine *View* initialisiert, existiert auch die *ConvertView* und hat die Eigenschaften der initialisierten View. Die *ConvertView* existiert während der gesamten Lebenszeit des Adapters und kann auf die eigentliche View gecastet (umgewandelt) werden. Dies bietet die Möglichkeit, das *noteListItem*-Objekt immer wieder zu recyceln, wie dies auch in der *if*-Anweisung erfolgt. Sie müssen hier unterscheiden zwischen den Daten-Objekten (*noteItem*) und Layout-Objekten (*noteListItemView*).

6.3.4 ListItem auswählen

Die *ListView* in unserer Anwendung macht nur Sinn, wenn bei einem kurzen Klick auf ein ListItem auch eine Aktion ausgeführt werden kann. Um auf einen Item-Klick reagieren zu können, wird das Interface *AdapterView.OnItemClickListener* der Klasse *android.widget.AdapterView* benötigt. Dieses Interface verlangt, dass die abstrakte Methode *onItemClick* implementiert wird. In dieser Methode wird das Klick-Ereignis eines ListItems behandelt. Weiterhin muss die *ListView* an den Ereignishandler gebunden werden. Der Methode *onItemClick* werden als Parameter die View und die Position des ausgewählten Items in der Liste übergeben.

1. Implementieren Sie das Interface *OnItemClickListener* durch `implements OnItemClickListener` in die Klasse *MainActivity*. Benutzen Sie die Code-Vervollständigung von Eclipse durch die Tastenkombination STRG + LEER. Die Signatur der Klasse: `public class MainActivity extends Activity implements OnItemClickListener{...}`.
2. Fügen Sie der Klasse die *onItemClick*-Methode der *MainActivity*-Klasse hinzu (Listing 6.11). Eclipse hat den Namen der Klasse rot markiert, und fordert Sie auf, nicht implementierte Methoden hinzuzufügen.
3. Erstellen Sie eine Klassenvariable *noteListView* vom Typ *ListView* in der *MainActivity* und ändern Sie die Bindung der ListView auf den neuen Adapter.
4. Fügen Sie folgenden Code in die *onCreate*-Methode der *MainActivity* ein: `noteListView.setOnItemClickListener(this);`

Listing 6.11 onItemClick-Methode MainActivity

```
@Override
public void onItemClick(AdapterView<?> parent, View view, int position,
    long itemId) {
    Note note = new Note();
    note = (Note) noteListView.getItemAtPosition(position);
    Log.d(TAG, "onItemClick: " + note.getSubject() + "|" + String.valueOf(position));
}
```

5. Vervollständigen Sie die notwendigen Imports, nutzen Sie die Tastenkombination STRG + O.

Erläuterung

Der Klick auf ein Item in der *ListView* bewirkt, dass die Methode *onItemClick()* ausgeführt wird. Damit aber die Methode überhaupt ausgeführt werden kann, müssen Sie das Interface *OnItemClickListener* von *android.widget.AdapterView* implementieren und die abstrakte Methode *onItemClick()* des Interfaces überschreiben.

Weiterhin muss die *ListView* den *Listener* registrieren, damit *callback*-Methode *onItemClick()* überhaupt aufgerufen wird. Dies erfolgt mit der *ListView*-Methode *.setOnItemClickListener(...)*, und diese erhält den *Listener* als Parameter. Dies ist in unserem Fall die *MainActivity* selbst, deshalb **this**.

Der Methode *onItemClick(...)* werden als Parameter *parent* des Typs *AdapterView*, *view* des Typs *View*, *position* des Typs *int* und *id* des Typs *long* übergeben.

- *parent*: Ist die *AdapterView*, die den *Listener* registriert hat, in unserem Fall *listView*
- *view*: Ist die *ItemView* im Adapter, auf die geklickt wurde
- *position*: Ist der nullbasierte Index im Adapter
- *id*: Ist die *row id* des *items*

Da die *ListView* Note-Objekte enthält, und die Position in der Liste bekannt ist, wird mit *note = (Note) noteListView.getItemAtPosition(position);* auf das Notiz-Objekt in der *onItemClick*-Methode zugegriffen.

6.3.5 LogCat verwenden

Bis jetzt haben Sie keine Informationen, wie sich die App zur Laufzeit verhält. Es könnten Fehler in der Anwendung sein, die diese nicht zum Absturz bringen, oder Sie wollen nur Informationen an einer bestimmten Stelle des Programms, ohne aufwändig Schritt für Schritt die Anwendung im Debug-Modus auszuführen. Für diesen Zweck eignet sich hervorragend *LogCat*. *LogCat* ist ein System-Logger, der auf jedem Android-System immer aktiv ist. Dieser Logger schreibt permanent Meldungen in die *Log*-Datei, die in ihrer Größe beschränkt ist, um nicht das Datei-System zu füllen. Jede Anwendung oder jeder Prozess besitzt die Möglichkeit, *LogCat*-Meldungen zu übergeben. Wir wollen diese Möglichkeit nutzen, um herauszufinden, auf welches *Listitem* nun geklickt wurde. Eclipse, mit dem ADT-Plug-in, stellt uns hier eine *LogCat-View* zur Verfügung, in der die Meldungen zur Laufzeit ausgegeben werden, alle Systemmeldungen oder gefiltert nach der Anwendung.

Die Klasse *Log* aus dem Paket *android.util* ermöglicht zur Laufzeit die Ausgabe von Informationen. Welche Informationen Sie ausgeben, bleibt Ihnen überlassen. In Listing 6.11 ist der Einsatz von *Log* demonstriert. *Log* bietet einige Methoden zur Ausgabe von Meldungen an.

Die in Listing 6.11 verwendete Variante *Log.d(...)* erwartet als Parameter einen TAG und die Message. Diese Informationen werden im Debug-Level der LogCat-View gefiltert. Als TAG deklariert man meist den Namen der Klasse, und die Message ist frei definierbar.

Die Log-Message aus Listing 6.11 gibt als TAG = *MainActivity* und als Message = *onItemClick: subject* des Items | *position* aus. Der TAG ist als Klassenvariable `private final static String TAG = "MainActivity";` in der *MainActivity* deklariert. In Bild 6.4 sehen Sie die Ausgabe von *Log.d*.

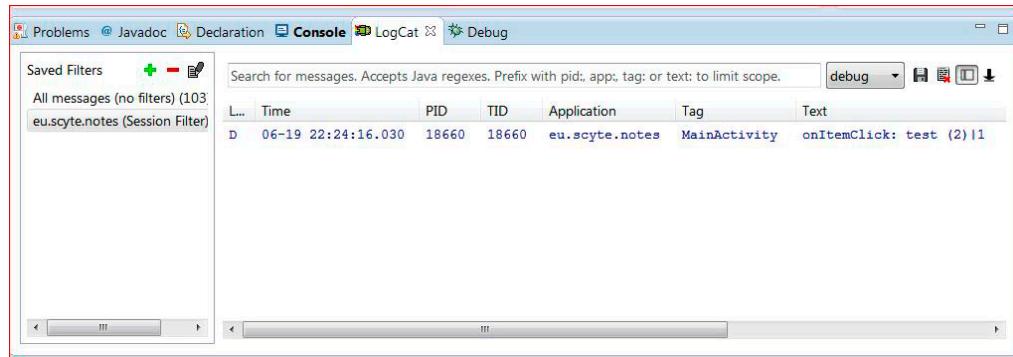


Bild 6.4 LogCat-View mit Ausgabe der Log-Meldung onItemClick-Methode



HINWEIS: Vor der Veröffentlichung einer Anwendung im Play Store müssen alle *Log.d*-Zeilen aus dem Code entfernt werden.

■ 6.4 Mit Dateien arbeiten

In diesem Abschnitt erfahren Sie nun endlich, wie eine Datei im Dateisystem gespeichert wird. Das Speichern von Dateien basiert in vielen Bereichen auf Dateioperationen aus dem Java-Paket *java.io*.

Wir wollen zunächst die Activity *NoteTextActivity-Klasse* so erweitern, dass eine Text-Notiz im Dateisystem gespeichert wird. Dazu muss dem Layout ein neues Edit-Text-Element *subject* (für den Dateinamen) hinzugefügt werden. Die Activity benötigt zudem noch einen Menüeintrag zum Speichern der Notiz. Alle Dateioperationen sollen in einer eigenen Klasse *FileHelper* erstellt werden. Diese *FileHelper*-Klasse wird bis zum Abschluss des Projekts alle notwendigen Methoden zum Speichern, Löschen, zur Namensermittlung usw. bereitstellen.

6.4.1 Datei im Dateisystem speichern

Da jede Anwendung unter Android in einer Sandbox ausgeführt wird, kann sie natürlich nicht einfach irgendwo gespeichert werden. Jeder Anwendung wird von Android ein eigener Speicherbereich zugewiesen. Dieses Verhalten übernimmt Android von seinem Linux-Kern.

Neben dem internen Speicher können Anwendungen die Daten auch im externen Speicher ablegen, jedoch muss die Anwendung die dafür notwendigen Berechtigungen besitzen. Allerdings sollten Sie beachten, dass abhängig vom Gerät mehrere externe Speicher vorhanden sein können. So hat Samsung bei seinen Galaxy-Modellen z. B. eine SD-Card fest in das System integriert. Dieser Speicher wird vom System als externer Speicher behandelt und als SD-Karte im Mount-Verzeichnis angezeigt. Bei der Desire-Serie von HTC hingegen erscheint nur die physikalische Erweiterung der SD-Karte als externer Speicherbereich im Mount-Verzeichnis.

1. Erstellen Sie ein neues Edit-Text-Element im Layout der *NoteTextActivity* und vergeben Sie die Id *subject*.
2. Erstellen Sie ein Menü für die Notizen. Geben Sie der XML-Datei den Namen *note.xml*. Dieses Menü soll auch für die Bild- und Sprach-Notizen genutzt werden.
3. Ändern Sie den Eintrag der *NoteTextActivity* im *AndroidManifest*, damit beim Starten der Activity automatisch das Soft-Keyboard angezeigt wird.
4. Erstellen Sie nun die Klasse *FileHelper*.
5. Jetzt fügen die Methode zum Speichern der Dateien, Listing 6.12, in die Klasse *FileHelper* ein.
6. Referenzieren Sie die *TextView*-Elemente für *Subject* und *Content* in der Klasse *NoteTextActivity*.
7. Erstellen Sie die private Methode *saveFile* innerhalb der Klasse *NoteTextActivity* zum Speichern der Text-Notiz.

Listing 6.12 Methode saveTextFile der Klasse FileHelper

```
public File saveTextFile(Context context, String subject,
                        String content,
                        File oldFile) throws IOException {
    File file;
    String rootPath;
    File externalStorage = Environment.getExternalStorageDirectory();
    String externalPath = externalStorage.getAbsolutePath()
        .replace("/mnt", "");
    externalStorage = new File(externalPath);
    File textDir = null;
    if(externalStorage.canWrite()){
        rootPath = externalStorage.getPath().toString();
        textDir = new File(rootPath + "/ScyteNotes/txt");
    } else if (context.getFilesDir().canWrite()){
        rootPath = context.getFilesDir().toString();
        textDir = new File(rootPath + "/txt");
    }
    String fileName = subject + ".txt";
    if(!textDir.exists()){
        textDir.mkdirs();
    }
}
```

```

textDir.setWritable(true);
textDir.setReadable(true);
file = new File(textDir, fileName);
if (oldFile == null) {
    if (file.exists()) {
        file = new File(
            checkFilePath(file.getAbsolutePath(), 0, ".txt"));
    }
} else if(oldFile.getAbsolutePath()
          .toString().startsWith("/mnt")){
    oldFile = new File(oldFile.getAbsolutePath()
                      .toString().replace("/mnt", ""));
} else if(!file.getAbsolutePath().toString()
          .equals(oldFile.getAbsolutePath().toString())) {
    if (file.exists()) {
        file = new File(
            checkFilePath(file.getAbsolutePath(), 0, ".txt"));
    }
    oldFile.renameTo(file);
}
try {
    FileOutputStream fileOutStr = new FileOutputStream(file);
    OutputStreamWriter outputStrWr = new OutputStreamWriter(
        fileOutStr,
        Charset.defaultCharset());
    BufferedWriter buffWriter = new BufferedWriter(outputStrWr);
    buffWriter.write(content);
    buffWriter.close();
} catch (Throwable t) {
}
}
}

```

Listing 6.13 Methode saveFile der Klasse NoteTextActivity

```

private void saveFile(){
    FileHelper fileHelper = new FileHelper();
    try{
        fileHelper.saveTextFile(this
                               , editText_Subject.getText().toString()
                               , editText_Content.getText().toString());
    } catch(IOException e){
        e.printStackTrace();
        Log.e(TAG, e.getMessage());
    }
}

```

Erläuterung

Die Klasse *FileHelper* ist eine Hilfsklasse, um Dateioperationen bereitzustellen. In dieser Klasse werden die Methoden zum Speichern, Löschen und der Namensermittlung bereitgestellt. Zunächst betrachten wir die Methode *saveTextFile(...)* der Klasse *FileHelper* (Listing 6.12). Die Speicher-Methode erwartet vier Parameter, den *context* des Typs *Context*, den *subject* und *content* jeweils des Typs *String* und *oldFile* vom Typ *File*. Mit *context* wird der Rahmen in der die Methode ausgeführt und übergeben. Aus dem *subject* wird der Dateiname erzeugt, und *content* ist der Inhalt der Datei. Mit dem Parameter *oldFile* erhält die

Methode das File-Objekt einer bestehenden Textdatei wenn diese geändert wurde. Weiterhin ist noch `throws IOException` im Methoden-Kopf deklariert. Eclipse erzwingt praktisch diese Implementierung, und `throws` kennzeichnet die Methode so, dass eine Ausnahme auftreten kann. Danach folgt der Typ `IOException`. Durch `throws` wird die Exception-Meldung an die aufrufende Methode weitergeleitet. Eine `IOException` kann bei Dateioperationen häufig auftreten, z.B. Datei wird nicht gefunden, Datei kann nicht gespeichert werden, da kein Speicherplatz mehr verfügbar ist etc.

Zunächst wird mit `Environment.getExternalStorageDirectory()` der Speicherpfad zum externen Speicher ermittelt. Die Klasse `Environment` aus dem Paket `android.os.Environment` stellt Informationen zu den Umgebungsvariablen des Systems zur Verfügung. Neben Informationen zum externen Speicherverzeichnis können Sie unter anderem auch Informationen zum Download- oder Mediaverzeichnis und deren Status abfragen. Der Rückgabewert der Methode enthält auf einigen Systemen den Pfad „/mnt“ des Mount-Verzeichnisses, das durch die String-Methode `replace()` entfernt wird. Dieser Teil des Pfades muss entfernt werden, da sonst die Methoden der Klasse `File` Fehler verursachen. In der Variable `externalStorage` erfolgt nun die Zwischenspeicherung des Pfades zum Root-Verzeichnis des externen Speichers, d.h., der extern oder intern vorhandenen SD-Karte.

In der folgenden `if`-Anweisung `if(externalStorage.canWrite())` wird überprüft, ob in das Root-Verzeichnis geschrieben werden kann. Ist das Ergebnis `true`, erfolgt die Erstellung des Root-Speicherpfades (`rootPath`) zum externen Speicher. In der folgenden Anweisung erfolgt nun mit `textDir = new File(new File(rootPath + "/ScyteNotes/txt");` die Erstellung des Speicherpfades, inklusive des Namens der App und des Unterverzeichnisses zur Ablage von Textnotizen.

Ist das Ergebnis der Überprüfung `false`, wird geprüft, ob im internen Speicher des Dateisystems der Anwendung (`context.getFilesDir().canWrite()`) geschrieben werden kann. Es handelt sich um das Verzeichnis `Files` im Datenverzeichnis der Anwendung. Wie Sie schon wissen, wird jede Android-Anwendung in einer Sandbox ausgeführt und erhält ihren eigenen Speicherbereich im internen Dateisystem. Ist das Ergebnis der Überprüfung `true`, erfolgt die Ermittlung des Pfades zur Anwendung im internen Speicher. Hier muss das Anwendungsverzeichnis nicht erstellt werden, weshalb dem Pfad nur das Unterverzeichnis für Textnotizen übergeben wird.

Die Klasse `File` aus dem Paket `java.io` bietet viele Methoden für die Verwaltung von Dateien. Sie sollten sich diese unbedingt unter <http://developer.android.com/reference/java/io/File.html> ansehen.

Mit der Anweisung `if(!textDir.exists())` wird geprüft, ob das Verzeichnis zum Speichern der Textnotizen schon besteht. Wenn dieses noch nicht vorhanden ist, wird es mit `textDir.mkdirs()` erstellt. Notwendige übergeordnete Ordner werden automatisch mit erstellt. Die Methode liefert bei Erfolg `true` zurück – und `false`, wenn das Verzeichnis schon existiert.

Der Dateiname wird mit `String fileName = subject + ".txt";` aus dem übergebenen Parameter `subject` mit Endung erstellt. Die Erstellung der Notiz-Datei erfolgt durch `File file = new File(textDir, fileName);`. Der erste Parameter ist das Datei-Objekt, welches das Verzeichnis repräsentiert, der zweite Parameter ist der String des Dateinamens mit Endung.

Im `try/catch`-Block erfolgt nun das Speichern der Datei. Dateien werden in Java durch Streams gespeichert. Als Erstes wird ein einfacher `FileOutputStream fileOutStr` erzeugt. Dieser Stream erhält als Parameter das Ziel, das Datei-Objekt der Notiz-Datei. Es würde

ausreichen, den *FileOutputStream* zum Speichern zu nutzen, jedoch ist dieser Stream nur geeignet, um ein einzelnes Byte oder ein Byte-Array in eine Datei zu schreiben. Bei dem Erzeugen von Textdateien gäbe es immer Probleme. Um diese Probleme zu umgehen, wird der *FileOutputStream* nicht direkt gefüllt, sondern durch den *OutputStreamWriter*. Der *OutputStreamWriter* kann mit der *write*-Methode Strings richtig verarbeiten. Obwohl der *OutputStreamWriter* Text per Default in UTF-8 speichert, haben wir trotzdem noch einmal das *DefaultCharset* angegeben, dies ist UTF-8. Die Daten werden gepuffert an den *OutputStreamWriter* übergeben, um Ressourcen zu schonen. Mit *buffWriter.write(content)*; wird die Datei geschrieben, und zum Abschluss müssen Sie den *BufferedWriter* mit *buffWriter.close()*; schließen. Das Schließen des *BufferedWriter* bewirkt, dass *OutputStreamWriter* und *FileOutputStream* geschlossen werden.

Wenden wir uns nun der Methode aus Listing 6.13 zu. Als Erstes wird mit *FileHelper fileHelper = new FileHelper();* initialisiert und dann die *saveTextFile*-Methode mit den Parametern aufgerufen. Der Kontext ist wieder die Activity, dem *subject* wird mit *editText_Subject.getText().toString()* der im String konvertierte Inhalt des *EditText*-Elements *subject* der Activity übergeben. Analog erfolgt dies mit dem *content-EditText*-Element.

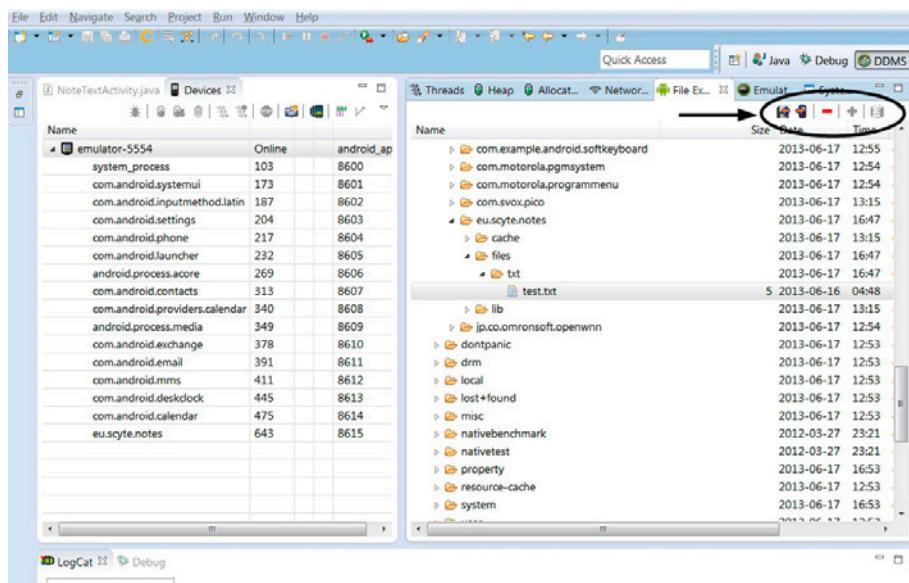
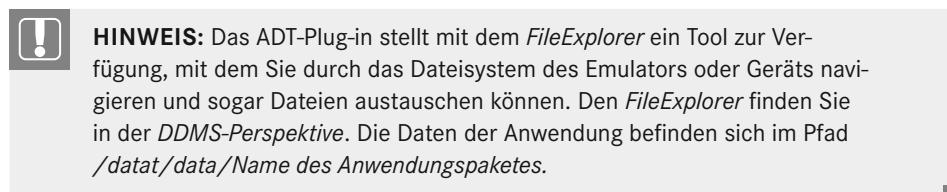
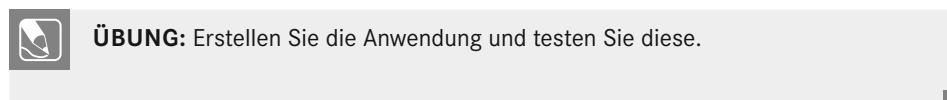


Bild 6.5 FileExplorer der DDMS-Perspektive in Eclipse

Jetzt wird die Datei im Dateisystem gespeichert. Allerdings ist dies so nicht in der App praktikabel. Es gilt nun, einige Probleme zu beseitigen. Beim Abspeichern einer Datei können drei Fälle eintreten:

1. Die Datei existiert noch nicht.
2. Die Datei existiert schon, und es wurde nur der Inhalt geändert. Der Betreff (Dateiname) bleibt gleich.
3. Die Datei existiert schon. Der Betreff (Dateiname) wurde geändert und der Inhalt gegebenfalls auch.

Außerdem gilt es zu beachten, dass beim Speichern einer neuen Datei überprüft werden muss, ob schon eine Datei mit diesem Namen existiert. Soll der Name einer Datei geändert werden, so entspricht dies dann einer neuen Datei, wobei zuerst die alte Datei umbenannt wird. Falls schon eine Datei mit diesem Namen existiert, soll der Speichervorgang nicht abgebrochen werden, sondern der Name der Datei um einen Zähler erweitert werden. Im Folgenden werden wir die Klasse *FileHelper* so erweitern, damit diese den Ansprüchen der Anwendung genügt. Der Rahmen ist mit der Methode *sayFile* bereits geschaffen.



Verwenden Sie bei Änderungen von schon genutzten Codes das Refactoring von Eclipse. Bei Änderungen werden automatisch alle Referenzen und Abhängigkeiten in anderen Code-Teilen angepasst: Markieren Sie die Variable, Methode und Klasse mit dem Cursor und führen RECHTS-KLICK + REFACTOR oder ALT + SHIFT + T aus.

Erweiterung der Klasse FileHelper

Zunächst wird die Methode zum Überprüfen des Dateinamens hinzugefügt. Diese benötigt weitere drei Hilfsmethoden. Aus Gründen der Einfachheit haben wir diese in einzelne Methoden gekapselt. Fügen Sie der Klasse *FileHelper* die Methoden aus Listing 6.14 hinzu.

Die Methode *checkFilePath()* wird von *saveTextFile()* mit dem zu überprüfenden, absoluten Dateipfad und dem Wert *count* aufgerufen, der beim Aufruf immer **0** sein muss. Falls der Wert von *count* **nicht größer 0** ist, wird mit der Methode *fileExists(...)* überprüft, ob die Datei bereits existiert. Ist dies der Fall, wird die Methode *checkFilePath()* wieder mit dem Dateipfad und um **1** erhöhten Wert von *count* **rekursiv** aufgerufen. Ansonsten wird einfach der Dateipfad zurückgegeben.

Ist die Datei schon vorhanden, wird ab dem nächsten Aufruf stets der erste *if*-Block ausgeführt. In diesem wird mit der Methode *deleteFileExtension()* zuerst die Dateierweiterung entfernt, dann entfernt *deleteFileNameCounter()* den im vorherigen Durchlauf angehängten *count*-Wert. Anschließend werden an den String ein Leerzeichen sowie der aktuelle *count*-Wert in Klammern angehängt, und mit der bereits beschriebenen Überprüfung auf eine bereits existierende Datei wird der Vorgang fortgeführt.

Das Ganze sieht so aus, dass, wenn die Methode für die Datei **test.txt** aufgerufen wird, und diese im Dateisystem schon existiert, im nächsten rekursiven Aufruf der Methode die Existenz der Datei **test (1).txt** überprüft wird. Anschließend wird die Datei **test (2).txt** überprüft und so weiter. Die Rekursion wird so lange durchgeführt, bis ein noch nicht existierender Dateipfad gefunden wurde, welcher dann mit *return filePath;* zurückgegeben wird.

Listing 6.14 Methoden zum Überprüfen des Dateinamens Klasse FileHelper

```
// Prüft, ob die Datei schon vorhanden ist.  
private boolean fileExists(String filePath) {  
    File file = new File(filePath);  
    if (file.exists()) {  
        return true;  
    } else {  
        return false;  
    }  
}  
  
// Diese Methode liefert den Datei-Pfad ohne Erweiterung zurück.  
public String deleteFileExtension(String filePath) {  
    filePath = filePath.substring(0, filePath.lastIndexOf("."));  
    return filePath;  
}  
  
// Diese Methode liefert den Datei-Pfad ohne Datei-Zähler zurück.  
public String deleteFileNameCounter(String filePath) {  
    if (filePath.lastIndexOf((char) 32) > 1) {  
        filePath = filePath.substring(0,  
            filePath.lastIndexOf((char) 40) - 1);  
    }  
    return filePath;  
}  
  
private String checkFilePath(String filePath  
    , int count  
    , String fileExtension) {  
  
    if (count > 0) {  
        filePath = deleteFileExtension(filePath);  
        filePath = deleteFileNameCounter(filePath);  
        filePath = filePath  
            + String.format(" (%d)", count)  
            + fileExtension;  
    }  
    if (fileExists(filePath)) {  
        count++;  
        return checkFilePath(filePath, count, fileExtension);  
    }  
    else {  
        return filePath;  
    }  
}
```

Weiterhin muss die Methode *saveTextFile* ergänzt und geändert werden.

Listing 6.15 Änderungen der Methode saveFileText der Klasse FileHelper

```
public File saveTextFile(Context context,  
                        String subject,  
                        String content,  
                        File oldFile) throws IOException {  
    .  
    .  
    .
```

```

File file = new File(textDir, fileName);
if (oldFile == null) {
    if (file.exists()) {
        file = new File(checkFilePath(
                            file.getAbsolutePath()
                            , 0
                            , ".txt"));
    }
} else if (!file.getAbsolutePath().toString()
           .equals(oldFile.getAbsolutePath().toString())) {
    if (file.exists()) {
        file = new File(checkFilePath(file.getAbsolutePath()
                                       , 0, ".txt"));
    }
    oldFile.renameTo(file);
}
try{...
    .
    .
    .
} catch (Throwable t) {
    t.printStackTrace();
}
return file;
}

```

Wie Sie bemerkt haben werden, ist die Methode nun vom Rückgabetyp *File* und enthält zusätzlich den Parameter *oldFile* des Typs *File*. Die Methode speichert weiterhin die Notiz im Dateisystem, allerdings werden einige Prüfungen durchgeführt.

Wir beginnen bei `if (oldFile == null)` mit der Betrachtung. Ist die Prüfung wahr, so entspricht dies der Erstellung einer neuen Datei. Im nächsten *if*-Block wird gefragt, ob die Datei schon existiert. Ist diese Bedingung auch wahr, so erfolgt die Ermittlung und Zuweisung eines neuen Dateinamens mit der eben erläuterten Methode *checkFilePath*. Trifft die Bedingung nicht zu, erfolgt keine Änderung des Namens, und die Datei wird gespeichert.

Wurde der Methode ein File-Objekt *oldFile* übergeben, so ist die Prüfung der Bedingung nicht wahr, und es geht im *else-if*-Block weiter. Zunächst wird geprüft, ob der absolute Pfad des File-Objektes *file*, der aus dem übergebenen *subject* erzeugt wurde, mit dem alten Pfad **nicht** übereinstimmt. Sind die Pfade unterschiedlich, prüft der folgende *if*-Block, ob die Datei schon existiert. Trifft dies zu, wird wieder ein File-Objekt mit gültigem Namen erzeugt. Am Ende des *else-if*-Blockes wird die bestehende Datei umbenannt, und der Speichervorgang wird ausgeführt. Zum Schluss gibt die Methode noch das gespeicherte File-Objekt zurück.

Um diesen Abschnitt abzuschließen, muss noch die Methode *saveFile* in der Klasse *NoteTextActivity* angepasst werden.

Listing 6.16 Änderungen der Methode *saveFile* der Klasse *NoteTextActivity*

```

public class NoteTextActivity extends Activity {

    File oldFile;

    private void saveFile() {

```

```
FileHelper fileHelper = new FileHelper();
try {
    oldFile = fileHelper.saveTextFile(this
        , editText_Subject.getText().toString()
        , editText_Content.getText().toString()
        ,oldfile);
} catch (IOException e) {
    e.printStackTrace();
}
editText_Subject.setText(fileHelper.deleteFileExtension(
    oldFile.getName().toString()));
}
```

Die Klassenvariable *oldFile* wird beim Starten der Activity mit **null** initialisiert. Beim erstmaligen Ausführen der *saveFile*-Methode wird **null** der *saveTextFile*-Methode der Klasse *FileHelper* übergeben. Nach erfolgreicher Ausführung nimmt *oldFile* das zurückgegebene File-Objekt entgegen. Am Ende der Methode wird der Text des *EditText-subject* aktualisiert. Die Aktualisierung des Textes erfolgt mit der *.setText()* von *EditText*. Innerhalb der *.setText*-Methode erfolgt der Aufruf der Methode *deleteFileExtension* der *FileHelper*-Klasse. Diese erhält den Dateinamen ohne Pfad als String-Parameter und liefert den Dateinamen ohne Dateierweiterung zurück. Der Dateiname wird durch die File-Methode *.getName* ermittelt und zur Sicherheit mit *.toString()* in String konvertiert.

6.4.2 Inhalte von Verzeichnissen auflisten und in ListView anzeigen

In den Abschnitten 6.3.3 und 6.4.1 wurde beschrieben, wie der Inhalt einer Liste in der *ListView* angezeigt und Dateien gespeichert werden. Jedoch werden die gespeicherten Dateien immer noch nicht in der *ListView* der *MainActivity* aufgelistet.

Die Methode *getAllFiles* aus Listing 6.17 erfüllt genau diese Aufgabe. Sie wird von der *Main Activity* aufgerufen und erhält als Parameter das Verzeichnis, aus dem alle Dateien und Verzeichnisse aufgelistet werden sollen. Diese werden durch die Methode *rootDir.listFiles()* in das File-Array *folderList* gespeichert. Die Methode *.listfiles()* ist eine Methode der Klasse *File*. Diese listet alle *File*-Objekte unterhalb des aktuellen Pfades auf. Diese Methode können Sie nur nutzen, wenn das *File*-Objekt ein Verzeichnis repräsentiert. Die erste *for*-Schleife geht der Reihe nach das File-Array durch, und in der *if*-Bedingung wird geprüft, ob das aktuelle *File*-Objekt ein Verzeichnis ist. Ist die Bedingung erfüllt, wird das Array *fileList* durch die Methode *.listFiles(fileFilter)* mit Dateien gefüllt. Die Methode erhält als Parameter den Dateifilter *fileFilter*. Dieser bewirkt, dass nur Dateien mit den Endungen **txt**, **png** oder **3gp** aus dem Verzeichnis gelesen werden. Jetzt durchläuft die zweite *for*-Schleife das Array *fileList* der Reihe nach und fügt die *File*-Objekte in die *ArrayList resultlist* ein. Ist der Vorgang beendet, wird die erste *for*-Schleife weiter ausgeführt, und der Vorgang wiederholt sich bis auch das Array *folderList* durchlaufen ist.

Nach Verlassen der *for*-Schleife wird die *ArrayList resultList* absteigend nach dem Datum sortiert. Um diese Sortierung zu erreichen, wurde die private Klasse *FileDateComparator* (Listing 6.18) erstellt, die das Interface *Comparator* aus dem Paket *java.util* implementiert. In der Klasse wurde ein Vergleichsalgorithmus erstellt, der Dateien absteigend nach Datum sortiert.

Der `Collections.sort`-Methode wird nicht nur die zu sortierende Liste übergeben, sondern auch noch ein Komparator. Die Sortermethode verwendet jetzt nicht mehr die eventuell vorhandene Standardmethode `compareTo()`, sondern vergleicht je zwei Elemente mit dem übergebenen Komparator.

Nachdem nun die Ergebnisliste auch sortiert ist, wird sie der aufrufenden Methode aus der `MainActivity` zurückgegeben.

Listing 6.17 Methode getAllFiles der Klasse FileHelper

```
public ArrayList<File> getAllFiles(File rootDir) {
    ArrayList<File> resultList = new ArrayList<File>();
    FileDateComparator comp = new FileDateComparator();
    FilenameFilter fileFilter = new FilenameFilter() {
        @Override
        public boolean accept(File f, String s) {
            return new File(f, s).isFile()
                && (s.toLowerCase().endsWith(".txt")
                    || s.toLowerCase().endsWith(".png")
                    || s.toLowerCase().endsWith(".3gp"));
        }
    }; // fileFilter
    try {
        File[] folderList = rootDir.listFiles();
        File[] fileList = null;
        for (int i = 0; i < folderList.length; i++) {
            if (folderList[i].isDirectory() == true) {
                fileList = folderList[i].listFiles(fileFilter);
                for (int a = 0; a < fileList.length; a++) {
                    resultList.add(fileList[a]);
                }
            } //if
        } // for
        Collections.sort(resultList, comp);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return resultList;
}
```

Listing 6.18 Klasse FileHelper der inneren Klasse FileDateComparator

```
private class FileDateComparator implements Comparator<File> {
    @Override
    public int compare(File f0, File f1) {
        if (f0.lastModified() < f1.lastModified()) {
            return 1;
        } else if (f0.lastModified() > f1.lastModified()) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

Um unsere Notizen in der *ListView* der *MainActivity* anzeigen zu können, müssen Sie noch einige Änderungen durchführen.

1. Die Klasse *Note* muss geändert werden.
2. Die Methode *setNoteItem* der Klasse *NoteListItemView* muss erweitert werden.
3. In der Klasse *MainActivity* muss die Methode *getFiles* erstellt werden.
4. In der *MainActivity* muss die *onResume*-Methode hinzugefügt und angepasst werden.
5. In der *MainActivity* muss der *ListAdapter* geändert werden.

Änderungen der Klasse Note

Die Klasse *Note* erhält nun einen Konstruktor `public Note (File file)` (Listing 6.19). Dem Konstruktor muss als Parameter ein File-Objekt übergeben werden. Wir verhindern somit, dass kein Note-Objekt mehr erstellt werden kann, für das keine Datei existiert. Innerhalb der Konstruktoren-Methode werden direkt die Attribute (Eigenschaften) einer Notiz, die sich von dem übergebenen File-Objekt ableiten lassen, zugewiesen. Einzige Ausnahme ist die Zuweisung des Typs einer Notiz. Der Übersicht wegen haben wir dies in der Setter-Methode *setNoteType()* gekapselt. Diese Methode ist *private* und kann nicht außerhalb der Klasse verwendet werden. Durch diese Vorgehensweise sind sofort alle Eigenschaften einer Notiz abrufbar, wie Sie gleich noch sehen werden.

Listing 6.19 Signatur und Konstruktor der Klasse Note

```
public class Note {  
    private String mSubject;  
    private String mFilePath;  
    private Long mFileDate;  
    private File mFile;  
    private NoteType mNoteType;  
    private ReminderType mReminderType;  
  
    public Note (File file) {  
        mFile = file;  
        String fileName = mFile.getName();  
        setNoteType(fileName);  
        mFileDate = (mFile.lastModified());  
        mSubject = fileName.substring(0, fileName.lastIndexOf("."));  
        mFilePath = mFile.getAbsolutePath();  
    }  
}
```

Listing 6.20 Methode *setNoteType (...)* der Klasse Note

```
private void setNoteType(String fileName) {  
    if (fileName.endsWith(".txt")) {  
        mNoteType = NoteType.text;  
    } else if (fileName.endsWith(".3gp")) {  
        mNoteType = NoteType.voice;  
    } else if (fileName.endsWith(".png")){  
        mNoteType = NoteType.sketch;  
    }  
}
```



Die Klasse *Note* repräsentiert viele Eigenschaften der Klasse *File*. Besser wäre es, eine neue Klasse, abgeleitet von der Klasse *File*, zu erstellen und um fehlende Eigenschaften zu implementieren. Vergleichen Sie einmal die Eigenschaften der Klasse *File* und der Klasse *Note*.

Änderungen der Klasse NoteListView

Um das Datum der Notiz anzuzeigen, muss die Methode erweitert werden. Die Methode erhält zusätzlich den Parameter *Context*.

Das *FileDate* des übergebenen *Note*-Objekts ist vom Daten-Typ *Long*, Millisekunden seit 1970, und muss erst in ein gültiges Zeitformat konvertiert werden. Die Konvertierung erfolgt in ein Datumsformat, das der eingestellten Sprache des Telefons entspricht. Mit `java.text.DateFormat df = android.text.format.DateFormat.getDateFormat(context.getApplicationContext());` wird ein *DateFormat*-Objekt erstellt, das die richtigen Formatinformationen enthält. Die Klasse `java.text.DateFormat` ist eine Klasse, die Datumswerte von Objekten des Typs *Date* formatiert. Die Klasse verlangt die Zuweisung des Formats als String, gültig wäre auch ein einfacher Format-String wie *dd.MM.yyyy*. Die Methode `getDateFormat(context.getApplicationContext())` liest das aktuell eingestellte Datumsformat aus dem System aus. Die Methode verlangt als Parameter einen Kontext. Hier wird der übergebenen Kontext benötigt, mit dem wir den Kontext der Anwendung ermittelt haben, der ja auch die Systemumgebung ist. Als Nächstes wird ein Datums-Objekt erstellt, dem das Datei-Datum der *Note* übergeben wird. Nun wird dem *EditText*-`modifyDate` ein ordentlich formatierter Datums-String mit `.setText(df.format(dt).toString())` zugewiesen.

Listing 6.21 Methode `setNoteItem()` der Klasse NoteListView

```
public void setNoteItem(Context context, Note note) {  
    subject.setText(note.getSubject());  
    java.text.DateFormat df =  
        android.text.format.DateFormat.getDateFormat(context.getApplicationContext());  
    Date dt = new Date(note.getFileDate());  
    modifyDate.setText(df.format(dt).toString());  
}
```

Änderung der MainActivity in der Methode `getFiles()`

Die Methode `getFiles()` wird in der `onResume()`-Methode aufgerufen. Als Erstes wird das Files-Verzeichnis der Anwendung ermittelt und dem File-Objekt *rootDir* übergeben. Als Nächstes wird eine Liste *files* erstellt, die in der Zuweisung die *FileHelper*-Methode `.getAllFiles` aufruft, als Parameter das Files-Verzeichnis übergibt und eine *ArrayList* mit den Dateien als Rückgabewert erhält. Danach wird die List *listNotes* bereinigt. Die *for*-Schleife durchläuft vollständig die Liste *files*, erstellt aus jedem File-Objekt ein Note-Objekt und fügt dieses in die Liste *listNotes* ein.

Listing 6.22 Methode getFiles() der Klasse MainActivity

```

private void getFiles() {
    String rootPath;
    try {
        File externalStorage = Environment
            .getExternalStorageDirectory();
        File rootDir = null;
        if(externalStorage.canWrite()){
            rootPath = externalStorage.getPath().toString();
            rootDir = new File(rootPath + "/ScyteNotes");
        } else if (this.getFilesDir().canWrite()){
            rootPath = this.getFilesDir().toString();
            rootDir = new File(rootPath);
        }
        List<File> files = new FileHelper().getAllFiles(rootDir);
        listNotes.clear();
        Note item;
        for (File file : files) {
            item = new Note(file);
            listNotes.add(item);
        }
        Log.d(TAG, "Files read end");
    }
}

```

Änderung der MainActivity in der onResume-Methode

Der Ablauf ist recht einfach. Sie werden sich allerdings fragen, warum extra die *onResume*-Methode der Activity gewählt wurde. Rufen Sie sich den LifeCycle einer Activity in Erinnerung und überlegen, wann die *onResume*-Methode aufgerufen wird. Wir haben diese Methode gewählt, da diese immer aufgerufen wird, wenn der User eine andere Activity verlässt, und diese wieder in den Vordergrund kommt. Somit wird die *ListView*-Ausgabe ohne viel Aufwand erneuert.

Nach Aufrufen der Methode liest die *getFiles*-Methode die Dateien neu ein. Der *ListAdapter* wird initialisiert, und es wird der **neue NoteListAdapter** verwendet, dem eine aktuelle Liste mit den Notizen, *listNotes*, übergeben wird. Als Letztes wird der Adapter wieder in der *List View* registriert.

Listing 6.23 Methode onResume() der MainActivity

```

@Override
protected void onResume() {
    super.onResume();
    getFiles();
    NoteListAdapter listAdapter = new NoteListAdapter(this, 0, listNotes);
    noteListView.setAdapter(listAdapter);
}

```

Änderung von MainActivity und ListAdapter in der onCreate()-Methode

Sie dürfen jetzt nicht vergessen, den alten *ListAdapter* aus der *onCreate()*-Methode zu entfernen. Löschen Sie diese beiden Zeilen:

- *ListAdapter listAdapter = new ArrayAdapter<Note>(this,...*
- *noteListView.setAdapter(listAdapter);*

6.4.3 Dateien löschen

Die aufrufende Methode übergibt der *deleteFileMethode* das zu löschen File-Objekt. Als Erstes wird überprüft, ob das File-Objekt existiert und beschrieben werden kann. Ist die Bedingung erfüllt, wird die Datei gelöscht, wenn nicht erfolgt keine Änderung.

Listing 6.24 Methode *deleteFile()* der Klasse *FileHelper*

```
public boolean deleteFile(File file) throws IOException {
    if (file.exists() && file.canWrite()) {
        file.delete();
    }
}
```

Damit eine Text-Notiz gelöscht werden kann, müssen Sie folgende Änderungen durchführen:

1. Fügen Sie im Menü der Notizen *note.xml* ein Item *delete* hinzu, erstellen Sie auch die entsprechende String-Ressource (Listing 6.25).
2. Erstellen Sie in der Klasse *NoteTextActivity* die Methode *deleteFile()* zum Löschen einer Datei (Listing 6.26).
3. Erweitern Sie die Methode *onOptionsItemSelected* der *NoteTextActivity* um das Menü-Item *delete*, das die Methode *deleteFile()* aufrufen soll (Listing 6.29). Vorher müssen Sie eine Klassevariable *menuDelete* des Typs *MenuItem* hinzufügen (Listing 6.27).
4. Erstellen Sie ebenfalls in der *NoteTextActivity* die Methode *onStart()*.
5. Löschen Sie *FileHelper fileHelper = new FileHelper()* aus der *saveFile()*-Methode der *NoteTextActivity*, erstellen Sie eine Klassenvariable *fileHelper* des Typs *FileHelper* und initialisieren Sie *fileHelper* in der *onStartMethode()* (Listing 6.30).
6. Fügen Sie in den *try*-Block der *saveFile()*-Methode folgende Zeile ein, *menuDelete.setEnabled(true);*.

Listing 6.25 delete-Item-Menü *note.xml*

```
<item android:id="@+id/note_menu_delete"
      android:title="@string/delete"
      android:icon="@android:drawable/ic_delete"
      android:showAsAction="never|withText">
</item>
```

Listing 6.26 Methode *deleteFile()* der Klasse *NoteTextActivity*

```
private void deleteFile() {
    if (oldFile != null) {
        try {
            fileHelper.deleteFile(oldFile);
            oldFile = null;
            menuDelete.setEnabled(false);
            editText_Subject.setText("");
            editText_Content.setText("");
        }
```

```
        editText_Subject.requestFocus();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Listing 6.27 Signatur und Klassenvariablen der Klasse NoteTextActivity

```
public class NoteTextActivity extends Activity {
    private static final String TAG = "NoteTextActivity";
    EditText editText_Subject;
    EditText editText_Content;
    File oldFile;
    FileHelper fileHelper;
    MenuItem menuDelete;
    ...
}
```

Listing 6.28 Methode onCreateOptionsMenu() der Klasse NoteTextActivity

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater menuInflater = getMenuInflater();
    menuInflater.inflate(R.menu.note, menu);
    menuDelete = menu.findItem(R.id.note_menu_delete);
    if (oldFile == null) {
        menuDelete.setEnabled(false);
    }
    return super.onCreateOptionsMenu(menu);
}
```

Listing 6.29 Methode onOptionsItemSelected() der Klasse NoteTextActivity

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.note_menu_save:
            Log.d("note text", "save note");
            saveFile();
            return true;
        case R.id.note_menu_delete:
            deleteFile();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Listing 6.30 Methode onStart() der Klasse NoteTextActivity

```
@Override
protected void onStart() {
    super.onStart();
    fileHelper = new FileHelper();
}
```

Erläuterung

Die `deleteFile()`-Methode darf nur aufgerufen werden, wenn auch ein File-Objekt existiert. Aufgerufen wird die Methode in der `onOptionsItemSelected()`-Methode, wenn das `MenuItem` vom Bediener betätigt wurde. Um zu verhindern, dass `deleteFile()` nach dem Start ausgeführt wird, wird es deaktiviert.

Direkt nach dem Start der Activity ist `oldFile = null`. Die Methode `onCreateOptionsMenu()` zur Erstellung des Menüs wird beim Starten der Activity ausgeführt. Zuerst wird das `note`-Menü durch den `MenuInflater` erstellt und die Referenz auf das MenuItem `menuDelete` angelegt. Im nächsten Schritt prüft die `if`-Anweisung, ob `oldFile` noch **null** ist. Trifft die Bedingung zu, erfolgt die Deaktivierung des MenuItems `menuDelete` durch die `boolsche` MenuItem-Methode `.setEnabled()`, die als Parameter **false** erhält.

Speichert ein User eine Notiz, wird die `saveFile`-Methode aufgerufen, und in dieser erhält `oldFile` eine Referenz auf ein File-Objekt im Dateisystem und das MenuItem `menuDelete` wird aktiviert. Da dies im `try`-Block der `saveFile`-Methode erfolgt, ist sichergestellt, dass ein File-Objekt zum Löschen existiert.

Wird `delete` im Menü betätigt, wird die Methode `deleteFile()` aus der `onOptionsItemSelected`-Methode aufgerufen. Als Erstes erfolgt noch einmal eine Prüfung, ob das `oldFile` ungleich **null** ist. Ist die Bedingung erfüllt, wird mit `fileHelper.deleteFile(oldFile)` die `deleteFile`-Methode der `FileHelper`-Klasse aufgerufen und erhält mit `oldFile` als Parameter die Referenz auf das File-Objekt im Dateisystem. Die `FileHelper`-Methode kann ausgeführt werden, da die Klasse `FileHelper` schon in der `onStart`-Methode referenziert ist. Im nächsten Schritt wird `oldFile` **null** und das MenuItem `menuDelete` wieder deaktiviert. Anschließend wird zuerst der Text des `EditText-subject`-Elementes entfernt und dann der Text des `EditText-content`-Elementes. Im Abschluss wird der Focus auf das `EditText-subject`-Element gesetzt.

6.4.4 Dateien lesen

Die Notizdatei soll automatisch gelesen werden, sobald der User in der Auswahlliste auf dem Startbildschirm einen Notizeintrag auswählt. Die Behandlung des `OnItemClick`-Ereignisses haben wir in Abschnitt 6.3.4 bereits behandelt. Außerdem verfügen Sie schon über Kenntnisse wie Intents verwendet werden. Der Aufruf der `readTextFile`-Methode erfolgt, wenn die `NoteTextActivity` durch einen expliziten Intent aus der `MainActivity` gestartet wurde und dieser Intent den absoluten Pfad zur Datei sendet.

Damit eine Textnotiz automatisch gelesen wird, müssen Sie folgende Änderungen durchführen:

1. Erweitern Sie die `onItemClick()`-Methode der Adapter-Klasse um den Aufruf der Methode `openNoteActivity(note);`.
2. Erstellen Sie in der `MainActivity` die Methode `openNoteActivity()` aus Listing 6.31.
3. Erweitern Sie die `onCreate()`-Methode der `NoteTextActivity` mit Listing 6.32.
4. Ergänzen Sie die Klasse `NoteTextActivity` um die Methode `readFile(File file)` (Listing 6.33), um eine Textnotiz lesen zu können.

5. Fügen Sie die Methode `readTextFile(File file)` (Listing 6.34) in die `FileHelper`-Klasse ein, um den Inhalt einer Textdatei als Ergebnis zu erhalten.

Listing 6.31 Methode zum Starten der Notizen-Activity und zum Senden des Dateipfades mit einem expliziten Intent aus der Klasse MainActivity

```
private void openNoteActivity(Note note){
    Intent intent = new Intent();
    switch (note.getNoteType()){
        case text:
            intent.setClass(getApplicationContext()
                , NoteTextActivity.class);
            intent.putExtra("filePath"
                , note.getFile().getAbsolutePath());
            startActivity(intent);
            break;
        case scetch:
            break;
        case voice:
            break;
        default: break;
    }
}
```

Listing 6.32 Erweiterung der `onCreate()`-Methode() der NoteTextActivity, Auswertung der vom Intent gesendeten Daten

```
final Bundle intentExtras = getIntent().getExtras();
if (intentExtras != null) {
    String filePath = null;
    try {
        filePath = intentExtras.getString("filePath");
        oldFile = new File(filePath);
        readFile(oldFile);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Listing 6.33 Methode zum Lesen der Textnotiz in der Klasse NoteTextActivity

```
private void readFile(File file) {
    if (file.isFile() && file.canRead()) {
        try {
            subject = fileHelper.deleteFileExtension(file.getName()
                .toString());
            editText_Content.setText(fileHelper.readTextFile(file));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Listing 6.34 Methode zum Lesen von Textdateien der Klasse FileHelper

```

public String readTextFile(File file) throws IOException {
    String resultStr = null;
    if (!file.canRead()) {
        return resultStr;
    }
    BufferedReader buffReader = null;
    try {
        FileInputStream fis = new FileInputStream(file);
        InputStreamReader inStreamReader = new InputStreamReader(fis,
            Charset.defaultCharset());
        buffReader = new BufferedReader(inStreamReader);
        StringBuilder sb = new StringBuilder();
        String line= "";
        while ((line = buffReader.readLine()) != null) {
            if(sb.length()>0){
                sb.append('\n');
            }
            sb.append(line);
        }
        resultStr = sb.toString();
    } catch (Throwable t) {
        t.printStackTrace();
    } finally {
        if (buffReader != null) {
            try {
                buffReader.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    return resultStr;
}

```

Erläuterung

Die Auswahl eines Items in der Liste des Startbildschirms bewirkt, dass die Methode *onItemClick()* ausgeführt wird. In der Methode erfolgt der Aufruf der *openNoteActivity(note)*, die als Parameter die Referenz auf das gewählte Notiz-Objekt enthält.

In der aufgerufenen Methode *openNoteActivity(note)* der *MainActivity* wird zunächst das Intent-Objekt *intent* erstellt. In der folgenden *switch/case*-Anweisung wird der Typ der Notiz ausgewertet und im entsprechenden *case*-Zweig des Notiztypen, im beschriebenen Fall der Text-Notiz, werden die Eigenschaften des Intents festgelegt. Zunächst erfolgt mit der *.setClass()*-Methode des Intents die Festlegung des Kontextes, in dem der Intent ausgeführt wird, und die Angabe der Klasse, die das Ziel des Intents ist. Mit der *.putExtra()*-Methode werden dem Intent die zu sendenden Daten übergeben. Der Versand der Daten erfolgt als Key/Value-Paar, damit diese in der Empfängerklasse identifiziert werden können. Der Key ist der eindeutige Identifizierer, Value sind die Nutzdaten. Durch Aufruf von *startActivity(intent)* wird nun die NoteTextActivity gestartet.

In der *onCreate()*-Methode der NoteTextActivity werden die gesendeten Daten des Intents ausgewertet. Ein Intent sendet Daten immer als Bundle, da mehrere Key/Value-Paare über-

mittelt werden können. Zunächst wird das **finale** Bundle-Objekt `intentExtras` erstellt. Das Bundle-Objekt muss final sein, weil die gesendeten Daten sich nicht mehr verändern. Mit der Anweisung `if (intentExtras != null)` wird überprüft, ob mit dem Intent Daten übermittelt wurden. Dies ist der Fall, wenn eine neue Notiz erstellt werden soll. Ergibt die Prüfung der `if`-Anweisung, dass das Bundle nicht **null** ist, wird der absolute Dateipfad der Notiz anhand des Keys ausgelesen. Der Klassenvariablen `oldFile` wird das Value des Keys als Wert zugewiesen und die Methode `readFile()` wird zum Lesen der Datei aufgerufen. Die `readFile()`-Methode erhält als Parameter die Referenz auf den Dateipfad.

In der `readFile()`-Methode der `NoteTextActivity` erfolgt zunächst die Prüfung, ob die übergebene Referenz eine Datei ist, und ob diese gelesen werden kann. Kann die Datei gelesen werden, wird der Klassenvariablen `subject`, die den Namen der Datei in der `ActionBar` darstellt, zunächst der Dateiname ohne Endung zugewiesen. Die Dateiendung wird durch die Methode `deleteFileExtension()` der `FileHelper`-Klasse entfernt. Als Parameter erhält die Methode den Namen der Datei, der durch die Klassenmethode `.getName()` der Klasse `File` ermittelt wurde.

Damit das `EditText`-Element den Inhalt der Datei anzeigt, wird in der `.setText()`-Methode der `EditText`-Klasse, die `readTextFile()`-Methode der `FileHelper`-Klasse aufgerufen.

Die `readTextFile()`-Methode der Klasse `FileHelper` erhält beim Aufruf als Parameter eine Referenz auf ein Dateiobjekt im Dateisystem und liefert als Ergebnis ein String-Objekt mit dem Inhalt der gelesenen Datei zurück.

Dann findet in der `if`-Anweisung mithilfe der Klassenmethode `.canRead()` der `File`-Klasse nochmals eine Überprüfung statt, ob das `File`-Objekt, also die Datei, gelesen werden kann. Ist das Ergebnis der Prüfung **false**, wird der Code innerhalb der geschweiften Klammern ausgeführt. `return resultStr` beendet die Ausführung der Methode und liefert als Ergebnis ein leeres String-Objekt zurück.

Um Textdateien lesen zu können, benötigen Sie wie in Abschnitt 6.4.1 sogenannte Streams. Zunächst wird ein `BufferedReader`-Objekt erzeugt. Im folgenden `try`-Zweig des `try/catch`-Blockes wird ein `InputStream` angelegt, der als Parameter die Referenz auf das einzulegende `File`-Objekt erhält. Der `InputStream` bietet grundlegende Funktionen, um Dateien zu lesen, jedoch ist dieser nur für das Lesen von binären Daten geeignet. Durch die Verkettung der Input-Streams erhalten wir am Ende ein String-Objekt in der richtigen Codierung. Der folgende `InputStreamReader` liest die Daten aus dem `InputStream` und konvertiert die Daten von Byte nach Char. Der `InputStreamReader` wird initialisiert, mit Verweis auf den `InputStream` und die Kodierung der Datei. Wird keine Kodierung im zweiten Parameter angegeben, verwendet Java standardmäßig 16-Bit-Unicode-Kodierung.

Um die Daten zeilenweise aus dem Stream lesen zu können, wird der `BufferedReader` verwendet. Der `BufferedReader` liest die vom `InputStreamReader` konvertierten Daten und puffert diese. Durch Verwendung des `BufferedReader` werden die Zugriffe auf das Dateisystem vermindert und die Geschwindigkeit beim Lesen der Dateien erhöht. Im oberen Teil der Methode wurde schon ein `BufferedReader`-Objekt angelegt. Dieses wird nun durch `new BufferedReader(inStreamReader)` initialisiert und erhält eine Referenz auf den konvertierten Stream des `InputStreamReader`.

Nachdem der `BufferedReader` initialisiert wurde, folgt die Erzeugung des String-Builders `sb` und des String-Objekts `line`. Das String-Objekt `line` erhält in der folgenden `while`-Schleife die

aktuell gelesene Zeile als Wert zugewiesen. Mit dem String-Builder werden die Zeilen zusammengefügt.

In der *while*-Schleife erfolgt das zeilenweise Lesen des Streams mit der *.readLine()*-Methode des *BufferedReader*. Da String-Objekte unveränderlich sind, kommt zum Erzeugen des Ergebnis-Strings die *StringBuilder*-Klasse zum Einsatz. Mit dem *StringBuilder* ist es möglich, Veränderungen am Objekt selbst vorzunehmen und diese anschließend in ein String-Objekt zu überführen. Mit der Methode *sb.append(line)* wird dem *StringBuilder*-Objekt der Inhalt der gelesenen Zeile angefügt. Die vorhergehende *if*-Anweisung überprüft die Länge des Strings *line*. Wenn sie ungleich **null** ist, wird ein Zeilenumbruch angefügt. Die *readline()*-Methode des *BufferedReader* wird solange ausgeführt, bis das Ende des Streams erreicht ist und keine Zeichen mehr gelesen werden. Da diese zeilenweise arbeitet, muss an die vorhergehende Zeile ein Zeilenumbruch angefügt werden, jedoch nicht am Ende. Deswegen erfolgt die Ausführung vor dem Anfügen einer neuen Zeile im *StringBuilder*-Objekt.

Nachdem alle Zeilen mit dem *BufferedReader* gelesen und die *while*-Schleife verlassen wurde, erfolgt mit der String-Methode *.toString()* die Wertzuweisung des *resultStr*.

Zum Abschluss erfolgt das Schließen der Streams im *finally*-Zweig. Es reicht jedoch aus, den *BufferedReader* zu schließen, da bei verketteten Streams nur das oberste Objekt geschlossen werden muss und somit automatisch alle untergeordneten Objekte geschlossen werden.

Nun erfolgt eine Überprüfung, ob der *BufferedReader* noch existiert. Ist dies der Fall, wird der Reader durch Aufruf von *buffReader.close()* geschlossen. Beim Schließen von Streams können Fehler auftreten. Eclipse erzwingt hier, dass die *.close()*-Methode des *BufferedReader* in einen *try/catch*-Block eingeschlossen wird. Auch die Verwendung der Streams kann nur in einem *try/catch*-Block erfolgen.

Als Ergebnis erfolgt jetzt mit *return resultStr* die Rückgabe des String-Objekts mit dem vom *BufferedReader* und *StringBuilder* erzeugten Inhalt der Datei.

■ 6.5 Zwischenstand der App (Version 0.3)

Nachdem wir in Kapitel 4 und 5 die grundlegenden Elemente der Oberfläche, die Verwendung von Ressourcen sowie die Erstellung von Optionsmenüs weitestgehend umgesetzt haben, zeigte ich Ihnen in diesem Kapitel, wie Sie mit Listen und Adapters umgehen, und wie Sie die Ergebnisse in einer ListView darstellen. Weiterhin zeigte ich Ihnen, wie Sie mit Hilfe von expliziten Intents eine weitere Activity starten, und wie Sie Daten an die neu gestartete Activity übergeben.

Außerdem erhielten Sie einen Einblick in die Verwendung des Android-Dateisystems und lernten, wie Sie unter Verwendung von Listen die Dateienamen aus Verzeichnissen in der ListView darstellen, und wie Sie Dateien speichern, löschen oder lesen.

7

ActionBar, WebView und E-Mail

In Kapitel 6 haben wir die Basis geschaffen, auf der Notizen erstellt, gespeichert, gelesen und gelöscht werden können.

In diesem Kapitel werden wir die ActionBar der Notizen-Activities anpassen, mithilfe von impliziten Intents E-Mails versenden und eine Hilfe- und About-Seite über die App anzeigen lassen.



Die Beispieldateien zu diesem Kapitel finden Sie unter
www.downloads.hanser.de im Unterordner *scytenotes 0.4*.

■ 7.1 ActionBar erweitern und Funktionen nutzen

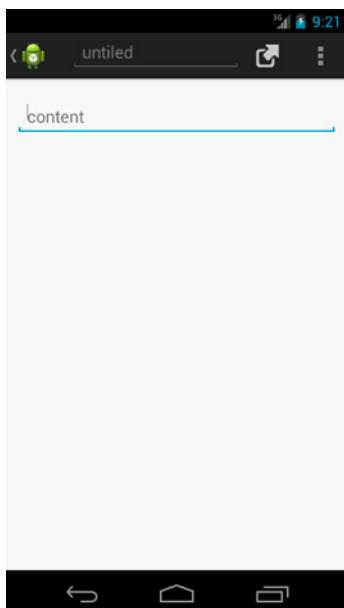
Die *ActionBar* in Android ist kein starres Gebilde. Sie kann entsprechend angepasst werden. In diesem Abschnitt erläutern wir, wie Sie die *ActionBar* auf Ihre Bedürfnisse anpassen und zeigen, wie Sie weitere Funktionen der *ActionBar* nutzen können. Im ersten Teil beschäftigen wir uns mit der Erstellung einer einheitlichen *ActionBar* für die Activities der Notizen, und im zweiten Teil wird die *ActionBar* der *MainActivity* angepasst.

7.1.1 Eigenes Layout-Element in der ActionBar verwenden und App-Icon für die Navigation nutzen

Die *ActionBar* soll in allen Activities gleich aussehen, um dem User eine konsistente Bedienung zu bieten. Außerdem soll bei einem Klick auf das Icon der Anwendung die Activity beendet und zur *MainActivity* zurückgekehrt werden.

1. Erstellen Sie eine Layout-Ressource *action_bar_notes.xml* und fügen Sie dem Layout ein *Edit Text*-Element hinzu.

2. Fügen Sie dem Notizen-Menü zwei neue Items hinzu und vergeben Sie die Id *note_menu_subject* und *note_menu_action*. Diese beiden Items sollen immer sichtbar sein. Dem *subject*-Item weisen Sie die String-Ressource *untitled* und dem *Action Layout* die *action_bar_notes.xml* zu. Dem *action-Item* wird eine neue String-Ressource *action* mit Wert *action* und dem Icon *ic_action_content_merge* zugewiesen.
3. Entfernen Sie das Edit Text-Element *noteText_editText_subject* aus dem Layout *activity_note_text.xml* und die Zeile *editText_subject = (EditText) findViewById(R.id.noteText_editText_subject);* aus der *onCreateMethode()*.
4. Passen Sie die NoteTextActivity (Bild 7.1) an und ändern Sie die *onCreate()*-Methode (Listing 7.3), die *onCreateOptionsMenu()*-Methode (Listing 7.4) und die *onOptionsItemSelected()*-Methode (Listing 7.5).

**Bild 7.1**

Screen ActionBar NoteTextActivity

Listing 7.1 Layout *action_bar_notes.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
    <EditText
        android:id="@+id/actionBar_editText_notes_subject"
        android:layout_width="180dp"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="false"
        android:layout_marginLeft="5dp"
        android:layout_marginRight="5dp"
        android:ems="10"
```

```
        android:hint="@string/untitled"
        android:imeOptions="actionNext"
        android:inputType="text"
        android:lines="1"
        android:maxLines="1" >
    </EditText>
</RelativeLayout>
```

Listing 7.2 Menu notes.xml, Item note_menu_subject und note_menu_action

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/note_menu_subject"
          android:title="@string/untitled"
          android:showAsAction="always"
          android:actionLayout="@layout/action_bar_notes" >
    </item>
    <item android:id="@+id/note_menu_action"
          android:showAsAction="always"
          android:icon="@drawable/ic_action_notesaction"
          android:title="@string/action">
    </item>
...
...
```

Listing 7.3 Methode onCreate(), Code der ActionBar

```
ActionBar actionBar = getSupportActionBar();
actionBar.setDisplayShowTitleEnabled(false);
actionBar.setDisplayHomeAsUpEnabled(true);
```

Listing 7.4 Methode onCreateOptionsMenu()

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater menuInflater = getMenuInflater();
    menuInflater.inflate(R.menu.note, menu);
    MenuItemSubject = (View) menu.findItem(R.id.note_menu_subject).getActionView();
    editText_Subject = (EditText) MenuItemSubject.findViewById(R.id.actionBar_editText_notes_subject);
    menuDelete = menu.findItem(R.id.note_menu_delete);
    if (oldFile == null) {
        editText_Subject.requestFocus();
        menuDelete.setEnabled(false);
    } else if (oldFile.canRead()) {
        readFile(oldFile);
        editText_Subject.setText(subject);
        editText_Content.requestFocus();
    }
    Log.d(TAG, "onCreateOptionsMenu() ");
    return super.onCreateOptionsMenu(menu);
}
```

Listing 7.5 Methode onOptionsItemSelected() mit neuen case-Anweisungen

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case android.R.id.home:  
            Intent intent = new Intent(this, MainActivity.class);  
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);  
            startActivity(intent);  
            return true;  
        case R.id.note_menu_action:  
            sendEmail();  
            return true;  
    ...
```

Erläuterung

Das erstellte neue Menü-Element wird in der *ActionBar* rechts ausgerichtet, da die *ActionBar* von rechts die Menü-Elemente hinzufügt. Dieses Verhalten können Sie mit einem eigenen Layout ändern, das Sie der *ActionBar* mit *.setCustomView()* zuweisen können.

Einige Attribute des *EditText*-Elements *actionBar_editText_notes_subject* sind zu beachten. Das Attribut *android:inputType="text"* sorgt dafür, dass das Keyboard das Layout zur Texteingabe bereitstellt. Das Attribut *android:imeOptions="actionNext"* ist verantwortlich dafür, dass bei Return im Keyboard das nächste Element im Layout den Focus erhält. In unserer Anwendung ist dies das *EditText*-content-Element. Mit *android:lines="1"* und *android:maxLines="1"* legen wir zur Sicherheit fest, dass nur eine Zeile angezeigt wird. Sie werden festgestellt haben, dass *android:hint="@string/untitled"* ein Text ist, der als Hinweis oder Anleitung gedacht ist. Achten Sie am Ende des Listings darauf, dass der Tag *<requestFocus />* entfernt ist, da wir den Focus auf die *EditText*-Elemente im Code steuern wollen.

Beim Starten der *NoteTextActivity* wird zuerst die *onCreate()*-Methode aufgerufen, , dann die *onStart()*, *onResume()* usw. aufgerufen und am Ende die *onCreateOptionsMenu()*-Methode. Deshalb kann die Erstellung der Referenz des *EditText* *subject*-Elements nicht in der *onCreate()*-Methode erfolgen, da diese vom Menu-Item *note_menu_subject* abhängig ist.

In der *onCreate()*-Methode wird durch *ActionBar actionBar = getActionBar();* die Referenz auf die *ActionBar* der Activity erstellt. Mit *.setDisplayShowTitleEnabled(false)*; wird die Anzeige des Titels, der Name der App, in der *ActionBar* deaktiviert. Die folgende Zeile *actionBar.setDisplayHomeAsUpEnabled(true);* bewirkt, dass vor dem Icon der Anwendung der Pfeil erscheint, und bei einem Klick auf das Icon die *onOptionsItemSelected()*-Methode mit der Item-Id *android.R.id.home* als Parameter aufgerufen wird.

In der *onOptionsItemSelected()*-Methode prüft die Switch/Case-Anweisung die übergebene Item-Id, und bei Übereinstimmung wird der Code im entsprechendem *case*-Zweig ausgeführt. Im *case*-Zweig des *android.R.id.home*-Items wird ein neuer expliziter Intent erzeugt, der die *MainActivity* als Empfänger adressiert. Das angehängte *Flag* bewirkt, dass das aktuelle Activity zum Löschen auf dem *Backstack* gekennzeichnet und geschlossen werden soll. Mit *startActivity(intent)* wird die *MainActivity* gestartet, die *NoteTextActivity* beendet und vom *Backstack* entfernt. Beim Betätigen des Zurück-Buttons in der Navigationsleiste des Telefons wird nicht mehr die *NoteTextActivity* angezeigt. Weitere Informationen zur Naviga-

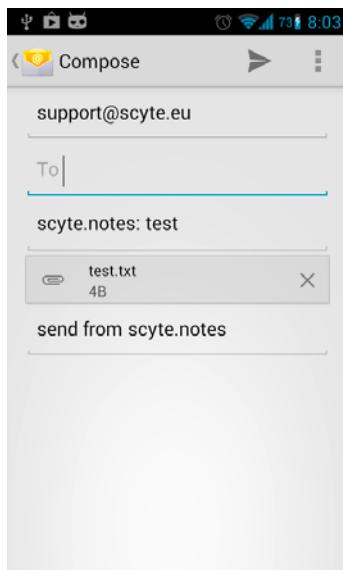
tion und zum *BackStack* finden Sie unter <http://developer.android.com/design/patterns/navigation.html>.

Die Änderungen in der *onCreateOptionsMenu()*-Methode der *NoteTextActivity* haben folgende Auswirkung. Nachdem der *MenuItemInflater* das *note*-Menu aufgefüllt hat, werden die notwendigen Referenzen erstellt. Zuerst wird eine Referenz auf das Menu-Item *note_menu_subject* erstellt, genauer auf die *ActionView* in der *ActionBar*. Die nächste Zeile `editText_Subject = (EditText) menuItemSubject.findViewById(R.id.actionBar_editText_notes_subject);` referenziert jetzt das *EditText*-Element, die eigentliche View, aus dem Layout *action_bar_notes.xml*. Diese Referenz wird allerdings nur richtig erstellt, da das Attribut *android:actionLayout="@layout/action_bar_notes"* des Menu-Items in der Menü-Ressource *note.xml* das Layout referenziert. Wäre die Referenz des *EditText*-Elements in der *onCreate()*-Methode erstellt worden, so wäre diese **null**. Im nächsten Schritt wir das Menü-Item *menuDelete* referenziert. Die folgende *if*-Anweisung überprüft nun, ob das File-Objekt *oldFile* **null** ist. *oldFile* kann ja nur eine Referenz aufweisen, wenn im Intent ein Dateipfad gesendet wurde. Ist die Bedingung erfüllt, erhält die *EditText*-View in der *ActionBar* den Focus, und das Menü-Element *menuDelete* wird deaktiviert. Im *else-if*-Zweig erfolgt die Überprüfung, ob das *File-Object oldFile* gelesen werden kann. Ist die Bedingung erfüllt, wird die Datei gelesen, und in der *readFile()*-Methode erhält die String-Variablen *subject* ihren Wert. Danach folgt `editText_Subject.setText(subject);`, und damit wird der Inhalt des Strings *subject* dem *Edit Text*-Element übergeben und zur Anzeige gebracht. Der Aufruf `editText_Content.requestFocus();` bewirkt nun, dass das *EditText*-Element des Notizen-Inhalts den Focus erhält. Das *return*-Statement liefert das Menü zurück.

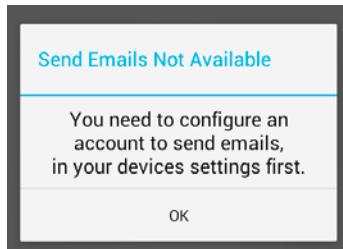
■ 7.2 E-Mail mit Anhang versenden

In diesem Abschnitt zeigen wir Ihnen, wie aus der App heraus die aktuelle Notiz als Anhang einer E-Mail versendet werden kann. Damit auch der aktuelle Inhalt der Notiz versendet wird, muss die Datei natürlich vorher gespeichert werden. In Wirklichkeit versendet nicht die Anwendung die E-Mail, sondern es wird ein impliziter Intent an das System gesendet, mit der Absicht, eine E-Mail zu senden. Das System entscheidet dann, welche Anwendung dies übernimmt und startet diese, z.B. GoogleMail wie in Bild 7.2. Sind mehrere E-Mail-Anwendungen installiert, erscheint ein Launcher, der die vorhandenen E-Mail-Apps zur Auswahl anbietet. Ist keine E-Mail-Anwendung vorhanden, kein Account eingerichtet, oder tritt ein anderer Fehler ein, erhält der User eine Hinweismeldung, wie Sie in Bild 7.3 zu sehen ist. Die Änderungen, die nötig sind, um eine E-Mail zu erstellen und dem System zum Versenden zu übergeben, sind sehr überschaubar.

1. Erstellen Sie als Erstes in der *NoteTextActivity* die Methode *sendEmail()* (Listing 7.7).
2. Fügen Sie der *onOptionsItemSelected()*-Methode der *NoteTextActivity* in der Switch/Case-Anweisung, im vorhergehenden Abschnitts erstellten *case*-Zweig des Action-Items, den Aufruf der *sendEmail()*-Methode hinzu (Listing 7.6).

**Bild 7.2**

GoogleMail mit erstellter E-Mail und Notiz im Anhang

**Bild 7.3**

Fehlermeldung im E-Mail-Versand

Listing 7.6 Aufruf der sendEmail()-Methode in der onOptionsItemSelected()-Methode

```
case R.id.note_menu_action:  
    sendEmail();  
    return true;  
...  
..
```

Listing 7.7 Methode sendEmail() der NoteTextActivity

```
public void sendEmail (){  
    saveFile();  
    if(oldFile !=null && oldFile.canRead()){  
        String email_subject = getResources().getString(R.string.email_subject) +  
subject;  
        String email_text = getResources().getString(R.string.email_text);  
        try {  
            Intent sendEmail = new Intent(Intent.ACTION_SEND);  
            sendEmail.putExtra(Intent.EXTRA_EMAIL, "");  
            sendEmail.putExtra(Intent.EXTRA_SUBJECT, email_subject);  
            sendEmail.putExtra(Intent.EXTRA_TEXT, email_text);  
            sendEmail.setType("plain/text");  
        }  
    }  
}
```

```
        sendEmail.putExtra(Intent.EXTRA_STREAM,
Uri.parse("file://" + oldFile.getAbsolutePath()));
        startActivity(sendEmail);
    } catch (Throwable t) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle(getResources().getString(R.string.send_email_alert_title));
        builder.setPositiveButton(R.string.ok, null);
        builder.setMessage(getResources().getString(R.string.send_email_alert_message));
        AlertDialog alertDialog = builder.show();
        TextView messageText = (TextView) alertDialog.findViewById(android.R.id.message);
        messageText.setGravity(Gravity.CENTER);
        alertDialog.show();
    }
} else{
    Log.d(TAG, "can't send email oldFile is null");
}
}
```

Erläuterung

Durch Betätigen des Action-Items in der *ActionBar* wird die Methode *sendEmail()* in der *onOptionsItemSelected()* Methode aufgerufen.

In der ersten Anweisung von *sendEmail()* erfolgt das Speichern der Notiz durch *saveFile()*. Im Folgenden wird der String *email_subject*, der später als Betreff der E-Mail verwendet wird, aus der String-Ressource *email_subject* und dem String *subject* (Titel der Notiz) zusammengesetzt. Die nächste Anweisung erstellt mit dem String *email_text*, den Text der E-Mail ebenfalls aus einer String-Ressource. Im *try*-Block wird zuerst der implizite Intent *send Email* mit der Absicht zum Versenden einer E-Mail erstellt. Festgelegt wird die Absichtserklärung in der Initialisierung des Intents durch *Intent.ACTION_SEND*. Die nächste Zeile *sendEmail.putExtra(Intent.EXTRA_EMAIL, "")*; fügt dem Intent die E-Mail-Adresse hinzu. Da wir den Empfänger noch nicht wissen, bzw. die Auswahl in der E-Mail-App sehr komfortabel ist, wird nichts übergeben. Die nächsten zwei Zeilen fügen den Betreff und den E-Mail-Text dem Intent hinzu. Der Typ der E-Mail, Plaintext oder HTML legt die Option *.setType("plain/text")* mit Text fest. Jetzt wir die Datei angehängt, besser der Intent erhält *Extra_STREAM* und eine im System auflösbare URI.

Am Ende des *try*-Blocks wird der Intent mit *startActivity(sendEmail)* an das System gesendet. Wenn alles gut geht und die Rahmenbedingungen stimmen erscheint die Email App mit Betreff, Text und angehängter Datei.

Schlägt die Aktion fehl wird der *catch*-Block aufgerufen. Da nicht bekannt ist was für ein Fehler vom System zurückgemeldet wird, ist der Typ des erwarteten Fehlers die oberste Klasse *Throwable*. In dem *catch*-Block haben wir diesmal eine eigene Fehlerbehandlung eingebaut. Der Grund ist das unterschiedliche Verhalten der einzelnen API-Level in denen die Anwendung ausgeführt werden kann, das Verhalten kann ein Absturz der Anwendung sein API-Level 14 oder eine vom System generierte Meldung. Ziel ist es jedoch dem User klar mitzuteilen das er eine Email-Anwendung mit konfiguriertem Konto benötigt.

Android verfügt über verschiedene Möglichkeiten dem User Hinweismeldungen anzugeben. Wir verwenden hier den *AlertDialog* da dieser sehr einfach und flexibel Konfigurierbar ist. Außerdem wollen wir sicherstellen, dass der User die Meldung wahrnimmt.

Zunächst wird im *catch-Block* `AlertDialog.Builder builder` initialisiert. Dieser Builder dient zur Erstellung des Dialog-Layouts. Mit `builder.setTitle(getResources())` und `builder.setMessage(getResources())` werden dem Dialog die Überschrift und die anzuzeigende Nachricht aus den String-Ressourcen hinzugefügt. In der nächsten Zeile erhält der Dialog mit `builder.setPositiveButton(...)` einen Positive Button, der den Text OK anzeigt und als String-Ressource definiert ist. Der zweite Parameter ist `null`, da wir nur einen Button OK haben, und wir keine Informationen benötigen, ob dieser betätigt wurde. Andernfalls müssten wir ein `DialogInterface` implementieren. Der Dialog schließt beim Klick auf OK. Mit `AlertDialog alertDialog = builder.create();` wird ein Dialog erstellt, aber noch nicht angezeigt. Jetzt ist die eigentliche Dialog-View im Hintergrund vorhanden, und wir können mit den bekannten Methoden auf die Standard-Elemente in der View zugreifen. Im nächsten Schritt wird mit `... alertDialog.findViewById(android.R.id.message);` eine Referenz auf die `TextView` der Meldung erstellt, um den Text mit der folgenden Anweisung `messageText.setGravity(Gravity.CENTER);` zentriert auszurichten. Mit `alertDialog.show();` wird der `AlertDialog` angezeigt.

■ 7.3 Lokale und externe Webseiten anzeigen

Wie schon beschrieben, nutzt die App eine lokale HTML-Seite, um die Hilfe und die About-Informationen anzuzeigen. In der *About*-Seite sind zwei Links vorhanden, einmal um die Webseite der Entwickler aufzurufen und um eine E-Mail zu versenden. In diesem Abschnitt zeigen wir Ihnen, wie eine lokale Webseite in der *WebView* angezeigt wird und wie mit URL-Aufrufen in der *WebView* umgegangen werden kann. Android versucht automatisch, eine URL bei einem Klick-Event mit dem Standard-Browser zu öffnen. Allerdings führt dies zu einem Fehler, wenn die URL keine Web-Adresse ist, sondern ein Link zum E-Mail-Versand oder eine Telefonnummer ist.

7.3.1 Erstellen und Anpassen der WebView Activity

In der App werden die Hilfe und die *About*-Informationen als lokale HTML-Dateien angezeigt. Dies erleichtert die Pflege und Erweiterung der Informationen. Es müssen nur noch die Dateien ausgetauscht werden. Weiterhin können Sie ganz einfach sprachabhängige Dateien zur Verfügung stellen. Am Beispiel der *About*-Informationen und Hilfe erläutern wir die Anzeige von HTML-Dateien.

Im Menü sind die Punkte zum Aufrufen der Hilfe und der App-Informationen schon vorhanden, jedoch haben diese noch keine Funktion. Die Hilfe und die App-Informationen sind nur einzelne HTML-Dateien, deshalb wird nur eine Activity benötigt, die HTML-Dateien anzeigt. Abhängig vom gewählten Menüeintrag *help* oder *about*, wird die entsprechende HTML-Datei in der Activity angezeigt (Bild 7.4).



Bild 7.4
about.html in der WebView

1. Zunächst erstellen Sie das Layout der Activity. Benötigt wird eine neue Layout-Ressource *activity_html.xml*. Fügen Sie dem Layout nur ein *WebView*-Element hinzu. Das *WebView*-Element füllt das ganze Layout, ist mittig und oben ausgerichtet. Die ID der *WebView* ist *webView*.
 2. Erstellen Sie zwei HTML-Dateien, *about.html* und *help.html*, beliebigen Inhalts, jedoch sollten Sie diese unterscheiden können. Fügen Sie zwei Links in die HTML-Dateien ein, einen zum Aufruf einer Web-Adresse (HTML-Code: `href="http://www.scyte.eu"`) und einen Link zum Versenden einer E-Mail (HTML-Code: `href="mailto:support@scyte.eu"`).
 3. Erstellen Sie im Ressourcen-Ordner */res* ein neues Unterverzeichnis */raw*.
 4. Kopieren Sie nun Ihre beiden HTML-Dateien in den neuen Ordner */res/raw*. Führen Sie den Kopiervorgang innerhalb von Eclipse aus. Die Dateien werden dann sofort vom Builder erkannt und der *R.java* hinzugefügt.
 5. Nun erstellen Sie die Activity-Klasse *HtmlActivity*, diese erbt von *android.app.Activity*. Ergänzen Sie den Code der *HtmlActivity* (Listing 7.10).
- Jetzt müssen Sie noch die *MainActivity* anpassen. Erstellen Sie zuerst die Methode zum Starten der *HtmlActivity*. Benennen Sie diese `startHtmlActivity(...)`, (Listing 7.9).
6. Damit auch die Menüeinträge *about* und *help* in der *MainActivity* ausgewertet werden, müssen Sie die Methode `onOptionsItemSelected(MenuItem item)` anpassen (Listing 7.8).
 7. Zum Abschluss dürfen Sie nicht vergessen, die *HtmlActivity* im *Android-Manifest* bekanntzumachen sowie den Versions-Code und den Versions-Namen zu ändern.

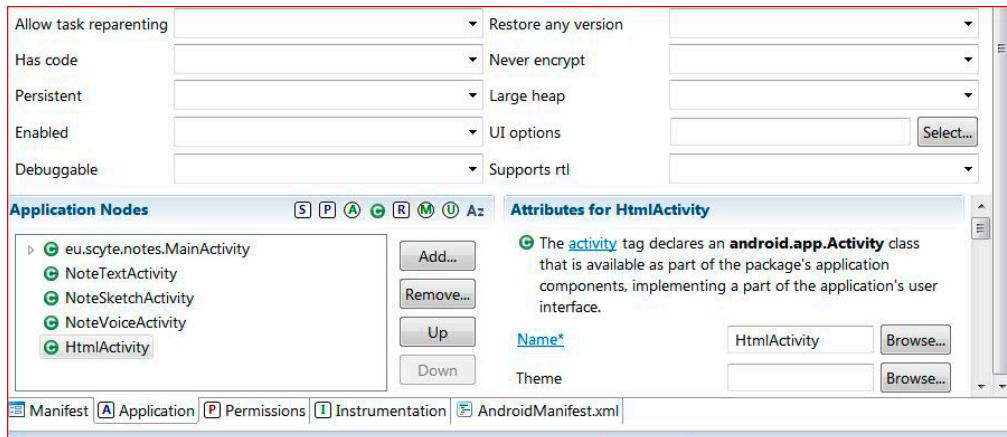


Bild 7.5 HtmlActivity im Android-Manifest

Erklärung

Bei einem Klick im Menü der *MainActivity* wird immer die Methode `onOptionsItemSelected(MenuItem item)` ausgeführt. Als Parameter erhält die Methode das Item, auf dem der Klick ausgeführt wurde. Die Switch/Case-Anweisung wertet die Item-Id aus, und bei Übereinstimmung wird der entsprechende *case*-Zweig aufgerufen. Im *case*-Zweig erfolgt der Aufruf der Methode `startHtmlActivity` mit einem String als Parameter.

Listing 7.8 MainActivity-Methode onOptionsItemSelected (MenuItem item)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.main_menu_add:
            startActivity(new Intent(this, NoteTextActivity.class));
            break;
        case R.id.main_menu_about:
            startHtmlActivity("about");
            return true;
        case R.id.main_menu_help:
            startHtmlActivity("help");
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

Die Methode `startHtmlActivity` startet die *HtmlActivity*. Der Start der *HtmlActivity* erfolgt durch einen expliziten Intent. Explizit, da dieser Intent genau die Klasse `HtmlActivity.class` verlangt. Mit diesem Intent wird der *HtmlActivity* durch `intent.putExtra("site", site);` der Parameterinhalt von *site* übermittelt. Die *HtmlActivity* wertet den Inhalt dann aus.

Die `onCreate`-Methode legt das Layout der Activity fest und referenziert die `WebView` des Layouts. Die Funktion `ActionBar` haben wir in Abschnitt 7.1.1 bereits erläutert.

Listing 7.9 MainActivity-Methode startet HtmlActivity(String site)

```
public void startHtmlActivity(String site){  
    Intent intent = new Intent(getApplicationContext(), HtmlActivity.class);  
    intent.putExtra("site", site);  
    startActivity(intent);  
}
```

Die durch den Intent gesendeten Daten werden in der Zeile `final Bundle intentExtras = getIntent().getExtras();` entgegengenommen. Da ein Intent mehrere Daten von Key/Value-Paaren, ein Bündel, übersenden kann, sind diese vom Typ *Bundle*. Dieses Bundle ist *final*, da es nach der ersten Zuweisung nicht mehr verändert werden darf. In der ersten *if*-Verzweigung (`intentExtras != null`) wird überprüft, ob Daten durch den Intent übermittelt wurden. In der zweiten *if*-Verzweigung (`intentExtras.getString("site").contentEquals("help")`) wird der Inhalt des Strings verglichen. Da die Werte als Key/Value-Paar vorliegen, muss der richtige Key gewählt werden. Bei Erfolg wird dem *String url* der entsprechende Pfad im Ressourcen-System übergeben. Im nachfolgenden *Try and Catch*-Block lädt die *WebView* die entsprechende lokale HTML-Seite. Der definierte Exception-Typ fängt nur Dokumenten-Fehler ab, also fehlerhafte Seiten.

Listing 7.10 HtmlActivity-Methode onCreate(Bundle)

```
public class HtmlActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_html);  
        ActionBar actionBar = getActionBar();  
        actionBar.setDisplayShowTitleEnabled(false);  
        actionBar.setDisplayHomeAsUpEnabled(true);  
        WebView webView = (WebView) findViewById(R.id.webView);  
        final Bundle intentExtras = getIntent().getExtras();  
        String url = null;  
        if (intentExtras != null) {  
            if (intentExtras.getString("site").contentEquals("help")) {  
                url = "file:///android_res/raw/help.html";  
            } else if (intentExtras.getString("site").contentEquals("about")) {  
                url = "file:///android_res/raw/about.html";  
            }  
            try {  
                webView.loadUrl(url);  
            } catch (LSEException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```



ÜBUNG: Starten Sie nun die App auf Ihrem Gerät oder Emulator und testen Sie die *HtmlActivity*. Rufen Sie die Links auf einem Gerät oder Emulator auf.

■ 7.4 Zwischenstand der App (Version 0.4)

In diesem Kapitel haben wir uns mit der Erweiterung und Nutzung von Funktionen der ActionBar, dem Erstellen und Versenden von E-Mails sowie dem Einsatz der WebView beschäftigt und diese Funktionalitäten in die Anwendung integriert. Weiterhin haben wir den AlertDialog und den AlertDialog.Builder eingesetzt, um eine Hinweismeldung anzugeben.

8

Fragments, Touch Events und Canvas

In Kapitel 7 haben wir die Beispiel-App zur Erstellung von Notizen um folgende Funktionalitäten erweitert:

- die ActionBar mit einem eigenen Layout-Element erweitern
- lokale oder externe Webseiten aufrufen
- E-Mails mit Anhang erstellen
- Hinweismeldungen in einem Alert Dialog anzeigen



Die Beispieldateien zu diesem Kapitel finden Sie unter
www.downloads.hanser.de im Unterordner *scytenotes 0.5*.

In diesem Kapitel zeigen wir Ihnen, wie Sie Fragments einsetzen, wie Sie Daten zwischen der Activity und dem Fragment austauschen, wie Sie mit Grafiken arbeiten, wie Sie auf die Galerie zugreifen und wie Sie ein Bild importieren. Ziel ist es, auf einem leeren oder importierten Bild Markierungen vorzunehmen oder darauf zu zeichnen.

■ 8.1 Fragments

Mit der Version ICS (4.0) von Android haben Fragments für alle Geräte Einzug gehalten. Unter Android gibt es verschiedene Typen von Fragments, die „normalen“ Fragments, das MapFragment zur Darstellung von Google Maps-Karten und das Dialog-Fragment, das zur Anzeige von Dialogen dient.

In diesem Abschnitt betrachten wird das normale Fragment ein wenig genauer und zeigen, wie das Fragment mit der Activity Daten austauschen kann.

8.1.1 Fragments, Drawable-Ressourcen

In der NoteSketchActivity und NoteVoiceActivity wird die untere Navigationsleiste als Fragment innerhalb des Layouts erstellt. Dies hat nur den Zweck, den Umgang und das Verständnis eines Fragments zu üben und zu festigen.

In diesem Abschnitt befassen wir uns mit dem Layout und der Kommunikation des Navigationsfragments, der NoteSketchActivity.

1. Kopieren Sie die Methoden `onCreateOptionsMenu()`, `onOptionsItemSelected()` und die Zeilen aus der `onCreate()`-Methode der NoteTextActivity in die `NoteSketchActivity`. Lassen Sie von Eclipse drei leere Methoden für `sendEmail()`, `saveFile()` und `deleteFile()` erstellen. Sie brauchen nur, nach dem Kopieren, den Cursor in die rot gestrichelten Methodenaufrufe der `onOptionsItemSelected()`-Methode setzen und die Tastenkombination STRG + 1 (Quick-Fix) ausführen. Als Nächstes erstellen Sie die Klassenvariablen `subject` des Typs `String`, `editText_subject` des Typs `EditText` und `menuDelete` des Typs `MenuItem`. Kommentieren Sie die `if/else`-Anweisung in der `onCreateOptionsMenu()`-Methode aus. Eclipse sollte nun keine Fehler mehr in der Klasse melden. Damit haben Sie das Layout und die meisten Funktionalitäten des Menüs und der ActionBar ebenfalls der NoteSketchActivity implementiert.
2. Zunächst erstellen Sie einige notwendige Ressourcen. Erweitern oder erstellen Sie die Dateien `dimens.xml`, `colors.xml` im Ordner `/res/values`. Weiterhin benötigen Sie im Ordner `/res/drawable` die Dateien der ToggleButtons `Color+_tggl_btn_.xml`, `Color+_tggl_btn_checked_shape.xml` und `Color+_tggl_btn_default_shape.xml` für die Farben `black`, `green` und `red`. Erstellen Sie für jeden Button diese drei Dateien.
3. Erstellen Sie jetzt die Layout-Ressource für die untere Bedienleiste der NoteSketch Activity. Erstellen Sie das Layout `action_bar_bottom_sketch.xml` (Listing 8.6). Ihr Ergebnis sollte aussehen wie Bild 8.1.

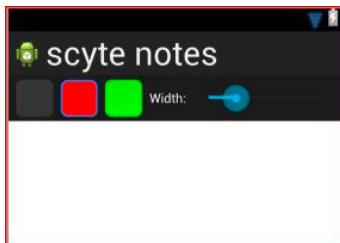


Bild 8.1

Layout-Ansicht `action_bar_bottom.xml`

4. Erstellen Sie die neue Java-Klasse `ActionBarBottomSketchFragment`, die von der Klasse `android.app.Fragment` erbt. Achten Sie darauf, dass Sie die richtige Oberklasse erweitern. Überschreiben Sie die Methode `View onCreateView()` mit dem Inhalt aus Listing 8.7.
5. Jetzt fügen Sie dem Layout der `NoteSketchActivity` ein Fragment hinzu, Bereich Layouts im Designer und richten es unten aus. Beim Hinzufügen erfolgt eine Abfrage nach der Fragment-Klasse, wählen Sie hier die in Schritt 4 erstellte Klasse aus. Ändern Sie die Attribute wie im Listing 8.8.

Listing 8.1 dimens.xml

```
<resources>
    <!-- ActionBar custom -->
    <dimen name="actionbar_height">48dip</dimen>
    <dimen name="actionbar_item_height">45dip</dimen>
    <dimen name="actionbar_item_width">45dip</dimen>
    <dimen name="button_item_height">40dip</dimen>
    <dimen name="button_item_width">40dip</dimen>
</resources>
```

Listing 8.2 colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="actionbar_bg_black">android:drawable/dark_header</color>
</resources>
```

Listing 8.3 black_tgl_btn.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:state_checked="true"
        android:drawable="@drawable/black_tgl_btn_checked_shape" />
    <item
        android:drawable="@drawable/black_tgl_btn_default_shape"/>
</selector>
```

Listing 8.4 black_tgl_btn_checked_shape.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <stroke
        android:width="2dp"
        android:color="#4876FF" />
    <gradient
        android:startColor="#363636"
        android:endColor="#3B3B3B"
        android:angle="90" />
    <corners
        android:topLeftRadius="7dp"
        android:topRightRadius="7dp"
        android:bottomLeftRadius="7dp"
        android:bottomRightRadius="7dp" />
</shape>
```

Listing 8.5 black_tgl_btn_default_shape.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#363636"
        android:endColor="#3B3B3B"
```

```
    android:angle="90" />
<corners
    android:topLeftRadius="7dp"
    android:topRightRadius="7dp"
    android:bottomLeftRadius="7dp"
    android:bottomRightRadius="7dp" />
</shape>
```

Listing 8.6 Layout action_bar_bottom_sketch.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="@dimen/actionbar_height"
    android:background="@color/actionbar_bg_black"
    android:gravity="center_vertical"
    android:orientation="horizontal" >

    <ToggleButton
        android:id="@+id/toggleButton_scetch_color_black"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="@dimen/button_item_width"
        android:layout_height="@dimen/button_item_height"
        android:layout_marginLeft="8dp"
        android:background="@drawable/black_tggl_btn"
        android:checked="false"
        android:onClick="onClick"
        android:textOff=""
        android:textOn="" />

    <ToggleButton
        android:id="@+id/toggleButton_scetch_color_red"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="@dimen/button_item_width"
        android:layout_height="@dimen/button_item_height"
        android:layout_marginLeft="8dp"
        android:background="@drawable/red_toggle_button"
        android:checked="true"
        android:onClick="onClick"
        android:textOff=""
        android:textOn="" />

    <ToggleButton
        android:id="@+id/toggleButton_scetch_color_green"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="@dimen/button_item_width"
        android:layout_height="@dimen/button_item_height"
        android:layout_marginLeft="8dp"
        android:background="@drawable/green_tggl_btn"
        android:checked="false"
        android:onClick="onClick"
        android:textOff=""
        android:textOn="" />

    <TextView
        android:id="@+id/textView_scetch_label_seekbar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
```

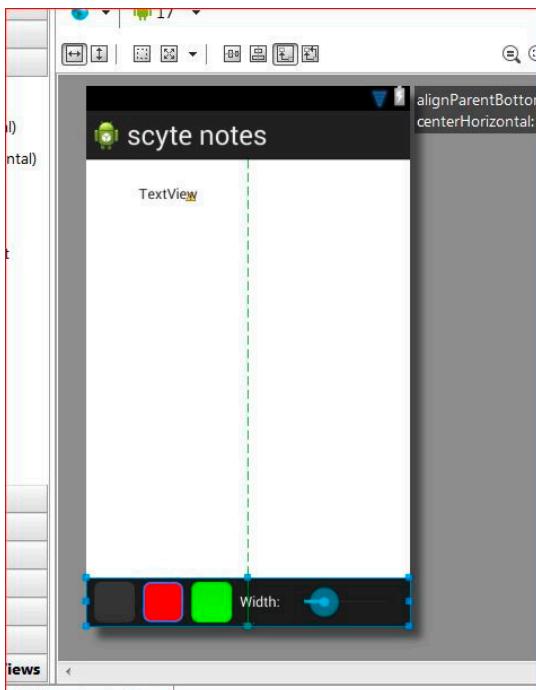
```
        android:layout_marginLeft="8dp"
        android:clickable="false"
        android:focusable="false"
        android:focusableInTouchMode="false"
        android:longClickable="false"
        android:text="@string/width"
        android:textColor="@android:color/primary_text_dark" />
<SeekBar
    android:id="@+id/strokeSeekBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:focusable="false"
    android:max="100"
    android:progress="25" />
</LinearLayout>
```

Listing 8.7 Klasse ActionBarBottomSketchFragment

```
public class ActionBarBottomSketchFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
    savedInstanceState) {
        View view = inflater.inflate(R.layout.action_bar_bottom_sketch, container, false);
        return view;
    }
}
```

Listing 8.8 Vorläufiges Layout activity_note_sketch.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/textView_subject"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="52dp"
        android:layout_marginTop="25dp"
        android:text="TextView" />
    <fragment
        android:id="@+id/action_bar_bottom_sketch"
        android:name="eu.scyte.notes.ActionBarBottomSketchFragment"
        android:layout_width="match_parent"
        android:layout_height="@dimen/actionbar_height"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        tools:layout="@layout/action_bar_bottom_sketch" />
</RelativeLayout>
```

**Bild 8.2**

Layout-Ansicht activity_note_sketch.xml

Erklärung Layout

Die Verwendung der Ressourcen *colors* und *dimens* sollte sich Ihnen sofort erschließen. In der *colors.xml* werden Farbressourcen und in der *dimens.xml* Dimensionen und Größenangaben definiert. Wie Sie am Beispiel der *colors.xml* sehen, müssen nicht unbedingt Werte direkt definiert werden. Es können auch Referenzen auf schon existierende Werteressourcen angelegt werden. Die Verwendung der neu angelegten Werteressourcen erfolgt analog zur Benutzung der String-Ressourcen, bei Attributen des entsprechenden Typs können diese zugewiesen werden. Der Designer schlägt diese im Drop-Down-Menü vor.

Die Drawable-Ressourcen der Toggle-Buttons ermöglichen nicht nur ein gefälliges Aussehen, sondern machen Umschalteffekte möglich. Ein Toggle-Button hat zusätzlich zu den Eigenschaften der normalen Buttons das Attribut des Zustandes *checked()*. Dieses Attribut verwenden wir, um abhängig von *checked()* das den Wert **true** oder **false** besitzt, dem Button eine andere Hintergrundressource zuzuweisen. Am Beispiel des roten Toggle-Buttons erläutern wir kurz das Verhalten. Sie haben schon in der *colors.xml* gesehen, dass in Ressourcen auf andere Ressourcen verwiesen werden kann. Der Toggle-Button hat im Layout für den Hintergrund die Ressource `android:background="@drawable/red_toggle_button"`. Die *red_toggle_button*-Ressource arbeitet ähnlich einer *switch/case*-Anweisung. Die Items in der Ressource werden der Reihenfolge nach durchlaufen, und bei entsprechender Übereinstimmung wird die Hintergrund-Ressource zugewiesen. Die Reihenfolge der Items ist in dieser Ressource von Bedeutung. Die Ressourcen *..._checked_shape* bzw. *..._default_shape* definieren das Erscheinungsbild des Buttons.

Die *SeekBar* können Sie einfach als eine Art Schieberegler betrachten. Diese hat ein Minimum und Maximum und liefert den aktuellen Wert zurück, wenn der Schieber verändert wird.

Beim Einfügen des Fragments wurden Sie nach der Klasse gefragt. Ihre Auswahl finden Sie im Wert des Attributes `android:name` wieder. Durch dieses Attribut wird im Layout die Referenz auf die Fragment-Klasse angelegt. Die `onCreateView()`-Methode in der Klasse des Fragments wird aufgerufen. In dieser wird durch den `Inflater` das Layout aufgefüllt. Dem Layout Inflater wird als erster Parameter das Layout übergeben. Die View Group ist das Layout, das aufgefüllt werden soll. Der Parameter `Bundle savedInstanceState` ist **false**, da keine Daten des Fragments persistiert werden sollen, wenn das Fragment verlassen wird. Fragments haben wie Activities einen Lebenszyklus. Müssen Daten beim Beenden des Fragments gesichert werden, so müssen Sie `savedInstanceState = true` übergeben und die Daten in der `onPause()`-Methode des Fragments persistieren. Es können nur Daten gesichert werden, die das Interface `Serializable` implementieren.

Am Ende liefert die `onCreateView()`-Methode die View des Fragments zurück, und das Activity-Layout bindet dieses ein.

Damit der Designer auch das Layout des Fragments vollständig in der Designansicht des Activity-Layouts anzeigt, muss der Namensraum der `Tools` im Layout angegeben werden, `xmlns:tools=http://schemas.android.com/tools`, und im Fragment müssen die Tools auf das Layout des Fragments verweisen, `tools:layout="@layout/action_bar_bottom_sketch"`.

8.1.1.1 Datenaustausch zwischen Fragment und Activity

Im weiteren Verlauf wollen wir uns zunächst um den Austausch von Daten zwischen dem Fragment und der Activity kümmern. Wird ein Button im Fragment betätigt, soll die gewählte Farbe an die Activity gesendet werden, und wenn der Regler der SeekBar verändert wird, soll die Activity den eingestellten Wert erhalten. Wir haben einen Wertebereich von 1 bis 100 definiert.

1. Initialisieren Sie die View-Elemente in der `OnCreateView()`-Methode.
2. Implementieren Sie in der Fragment-Klasse die Interfaces `OnCheckedChangeListener`, aus der Klasse `android.widget.CompoundButton`. und `OnSeekBarChangeListener`, aus der Klasse `android.widget.SeekBar`.
3. Fügen Sie die erforderlichen Methoden `onCheckedChanged()` des `OnCheckedChangeListener`-Interfaces, `onProgressChanged()`, `onStartTrackingTouch()` und `onStopTrackingTouch()` des `OnSeekBarChangeListener`-Interfaces in der Klasse `ActionBarBottomSketchFragment` ein.
4. Weisen Sie den Toggle-Buttons den `onCheckedChangeListener` mit `.setOnCheckedChangeListener(this)` des Interface zu.
5. Weisen Sie der `SeekBar` mit `strokeSeekBar.setOnSeekBarChangeListener(this);` `OnSeekBarChangeListener` in der `onCreateView()`-Methode zu.
6. Jetzt erstellen Sie die beiden Interfaces zur Übergabe der Daten aus dem Fragment an die Activity im Fragment. Es wird ein Interface für die Farbe und ein Interface für die Stärke der Linien benötigt (Listing 8.9 und Listing 8.10).
7. Nachdem Sie nun das Interface erstellt haben, ändern Sie die Methode `onCheckedChanged()` (Listing 8.11).
8. Passen Sie die Methoden der `SeekBar` an (Listing 8.12).

9. Registrieren Sie die Listener an der Activity in der *onAttach()*-Methode des Fragments (Listing 8.13).
10. Implementieren Sie das Interface *PaintColorListener* und *PaintStrokeListener* in der *NoteSketchActivity*-Klasse und fügen Sie dem Interface die Methoden hinzufügen (Listing 8.14).

Listing 8.9 Listener der Klasse und Interface PaintColorListener

```
private PaintColorListener myPaintColorListener;
private PaintStrokeListener myPaintStrokeListener;

public interface PaintColorListener{
    public void setPaintColor(int Color);
}
```

Listing 8.10 Interface PaintStrokeListener des Fragments

```
public interface PaintStrokeListener{
    public void setPaintStroke(int Stroke);
}
```

Listing 8.11 Methode *onCheckedChanged()* des Fragments

```
@Override
public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    Log.d(TAG, "checkedChanged");
    switch (buttonView.getId()){
        case R.id.toggleButton_scetch_color_black:
            if(isChecked){
                toggleButton_red.setChecked(false);
                toggleButton_green.setChecked(false);
                myPaintColorListener.setPaintColor(Color.BLACK);
                myPaintStrokeListener.setPaintStroke(10);
                strokeSeekBar.setProgress(10);
            }
            break;
        case R.id.toggleButton_scetch_color_red:
            if(isChecked){
                toggleButton_black.setChecked(false);
                toggleButton_green.setChecked(false);
                myPaintColorListener.setPaintColor(Color.RED);
            }
            break;
        case R.id.toggleButton_scetch_color_green:
            if(isChecked){
                toggleButton_black.setChecked(false);
                toggleButton_red.setChecked(false);
                myPaintColorListener.setPaintColor(Color.GREEN);
            }
            break;
        default:
            break;
    }
}
```

Listing 8.12 Überschreibende Methoden der SeekBar im Fragment

```

@Override
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    seekBarValue = progress;
    Log.d(TAG, "seekBar: " + String.valueOf(seekBarValue));
}

@Override
public void onStartTrackingTouch(SeekBar seekBar) {

}

@Override
public void onStopTrackingTouch(SeekBar seekBar) {
    if(seekBarValue < 1){
        seekBarValue = 1;
    }
    myPaintStrokeListener.setPaintStroke(seekBarValue);
}

```

Listing 8.13 Methode onAttach() des Fragments

```

@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    myPaintColorListener = (PaintColorListener) activity;
    myPaintStrokeListener = (PaintStrokeListener) activity;
}

```

Listing 8.14 Interface Methoden der NoteSketchActivity

```

@Override
public void setPaintColor(int color) {
    textView.setText(String.valueOf(color));
}

-----
@Override
public void setPaintStroke(int stroke) {
    textView.setText(String.valueOf(String.valueOf(stroke)));
}

```



ÜBUNG: Sie sollten jetzt den aktuellen Stand ausführen und erfolgreich testen.

Erklärung

Beim Betätigen eines Toggle-Buttons wird dessen Status auf `checked=true` geändert, Ausnahme dieser ist schon `true`. Da das Interface `OnCheckedChangeListener` implementiert ist und der Button an den Ereignishandler des Interface gebunden, wird die `onCheckedChange()`-Methode aufgerufen. In der Methode erfolgt wieder die Prüfung durch die `switch/case`-Anweisung, welcher Button das Ereignis sendet. Die `if`-Anweisung im `case`-Zweig des But-

tons überprüft den Status `checked=true`. Ist die Bedingung erfüllt, werden die nächsten Anweisungen ausgeführt. Die anderen Buttons werden auf `checked=false` gesetzt, und die letzte Anweisung im `case`-Zweig ruft den `myPaintColorListener` auf, der die `.setPaintColor Methode()` ausführt und als Parameter den Integer-Wert einer Farbe übergibt.

Die NoteSketchActivity implementiert die Interfaces `PaintColorListener` und `PaintStrokeListener` der Fragment-Klasse und ist gezwungen, die `public`-Methoden der Interfaces zu implementieren. In der `onAttach()`-Methode des Fragments wurden die Listener an der Activity angemeldet. Durch diese Registrierung der Listener in der Activity werden bei einem Ereignis, je nach Listener, die überschriebenen Methoden `setPaintColor()` oder `setPaintStroke()` in der `NoteSketchActivity` ausgeführt.

Die Implementierung des `OnSeekBarChangeListener` hat eine ähnliche Wirkung. Die Seek-Bar der Oberfläche des Fragments ist in der `onCreateView()`-Methode des Fragments referenziert und registriert den Listener des Interface durch `strokeSeekBar.setOnSeekBarChangeListener Listener(this);`. Das Interface erzwingt die Implementierung der drei Methoden `onProgressChanged()`, `onStartTrackingTouch()` und `onStopTrackingTouch()`, und daraus ergibt sich folgender Ablauf. Bei einem Touch auf den Regler der `SeekBar` wird zunächst die Methode `onStartTrackingTouch()` aufgerufen, die in unserer Anwendung keine Verwendung findet. Wird der Regler bewegt, wird die Methode `onProgressChanged()` ausgeführt und erhält den Wert des Reglers als Parameter `progress`. In der Methode wird der Klassenvariablen `seekBarValue` zugewiesen. Der Wertebereich ist im Layout der `SeekBar` durch Angabe von Max definiert, in unserem Fall ist er 0 bis 100.

Nimmt der User den Finger vom Regler, erfolgt der Aufruf der `onStopTrackingTouch()`-Methode. In dieser wird durch die `if`-Anweisung geprüft, ob der Wert kleiner 1 ist. Trifft dies zu, erhält `seekBarValue` den Wert 1. Im folgenden Schritt wird durch den `myPaintStrokeListener` die Interface-Methode `.setPaintStroke(seekBarValue)` ausgeführt und ruft damit die `setPaintStroke()` in die `NoteSketchActivity`, die den Wert von `seekBarValue` als Parameter entgegennimmt und der `TextView` diesen Wert als Text übergibt.

■ 8.2 Image View erweitern, Canvas und Touch Events

Die ImageView ist ein View Element, das für die Anzeige von Bildern geschaffen wurde. Da wir hier eine ganz kleine Art von Bildbearbeitung realisieren, lag es nahe, auf Basis dieser Klasse eine Grundlage zu schaffen, um auf Bildern zu zeichnen. Wer sich ein wenig mit Bildbearbeitung beschäftigt, der kennt das Folgende. Bei der Bildbearbeitung werden Elemente nicht direkt in das Bild gezeichnet, sondern, es wird eine Ebene über das Bild gelegt, auf der gezeichnet wird. Beim Speichern des Bildes werden die Bearbeitungsebene und das Bild zusammengefügt. In Android läuft dieser Mechanismus in ähnlicher Form ab. Der andere Punkt ist, dass wir einen Rahmen benötigen, zum Empfangen der Touch-Ereignisse auf dem Bildschirm, der die eigentliche Zeichenfläche beachtet.

1. Entfernen Sie die `TextView` aus dem Layout `activity_note_sketch.xml` und alle Referenzen in der `NoteSketchActivity`-Klasse.

2. Erstellen Sie als Nächstes die `DrawableImageView`-Klasse (Listing 8.15).
3. Fügen Sie dem Layout ein `ImageView`-Element hinzu, das Sie über dem Fragment platzieren (Listing 8.16).
4. Ergänzen Sie die `onCreate()`-Methode der `NoteSketchActivity` (Listing 8.17).

Listing 8.15 Klasse `DrawableImageView`

```
public class DrawableImageView extends ImageView {  
    private Bitmap mBitmap;  
    private Canvas mCanvas;  
    private Path mPath;  
    private Paint mBitmapPaint;  
    private Paint mPaint;  
  
    public DrawableImageView(Context context, AttributeSet attrs) {  
        super(context, attrs);  
        mPaint = new Paint();  
        mPaint.setAntiAlias(true);  
        mPaint.setDither(true);  
        mPaint.setColor(Color.RED);  
        mPaint.setAlpha(45);  
        mPaint.setStyle(Paint.Style.STROKE);  
        mPaint.setStrokeJoin(Paint.Join.ROUND);  
        mPaint.setStrokeCap(Paint.Cap.ROUND);  
        mPaint.setStrokeWidth(25);  
        mPath = new Path();  
        mBitmapPaint = new Paint(Paint.DITHER_FLAG);  
    }  
  
    protected void onDraw(Canvas canvas) {  
        canvas.drawBitmap(mBitmap, 0, 0, mBitmapPaint);  
        canvas.drawPath(mPath, mPaint);  
    }  
  
    @Override  
    public void setImageBitmap(Bitmap bm) {  
        super.setImageBitmap(bm);  
        Log.d(TAG, "setImageBitmap");  
        mBitmap = bm.copy(Bitmap.Config.ARGB_8888, true);  
        mCanvas = new Canvas(mBitmap);  
    }  
    // EigenschaftenDraw  
  
    public void setPaintStroke(int stroke) {  
        mPaint.setStrokeWidth(stroke);  
        Log.d(TAG, "setStroke: ");  
    }  
  
    public void setPaintColor(int Color) {  
        mPaint.setColor(Color);  
        if (Color == -16777216) {  
            mPaint.setDither(false);  
            Log.d(TAG, "setAlpha(100)");  
        }  
        else {  
  
            mPaint.setAlpha(45);  
            mPaint.setDither(true);  
        }  
    }  
}
```

```
        }
        Log.d(TAG, "setColor: " + String.valueOf(Color));
    }

    // Touch
    private float mX, mY;
    private static final float TOUCH_TOLERANCE = 4;

    private void touch_start(float x, float y) {
        mPath.reset();
        mPath.moveTo(x, y);
        mX = x;
        mY = y;
    }

    private void touch_move(float x, float y) {
        float dx = Math.abs(x - mX);
        float dy = Math.abs(y - mY);
        if (dx >= TOUCH_TOLERANCE || dy >= TOUCH_TOLERANCE) {
            mPath.quadTo(mX, mY, (x + mX) / 2, (y + mY) / 2);
            mX = x;
            mY = y;
        }
    }

    private void touch_up() {
        mPath.lineTo(mX, mY);
        // commit the path to our offscreen
        mCanvas.drawPath(mPath, mPaint);
        // kill this so we don't double draw
        mPath.reset();
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        float x = event.getX();
        float y = event.getY();

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                touch_start(x, y);
                invalidate();
                break;
            case MotionEvent.ACTION_MOVE:
                touch_move(x, y);
                invalidate();
                break;
            case MotionEvent.ACTION_UP:
                touch_up();
                invalidate();
                break;
        }
        return true;
    }
}
```

Listing 8.16 Neue ImageView im Layout activity_note_sketch.xml

```
<eu.scyte.notes.DrawableImageView
    android:id="@+id/drawableImageView"
    android:layout_width="fill_parent"
    android:layout_height="match_parent"
    android:layout_above="@+id/action_bar_bottom_sketch"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:adjustViewBounds="true"
    android:scaleType="fitCenter" />
```

Listing 8.17 Änderungen der NoteSketchActivity onCreate

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    display = getWindowManager().getDefaultDisplay();
    displayMetrics = new DisplayMetrics();
    display.getMetrics(displayMetrics);
    mHeight = displayMetrics.heightPixels;
    mWidth = displayMetrics.widthPixels;
    mDensity = displayMetrics.densityDpi;
    physDisplayPixel = displayMetrics.xdpi * displayMetrics.ydpi;
    drawableImageView = (DrawableImageView) findViewById(R.id.drawableImageView);
    setCanvasImage(null);
    fileHelper = new FileHelper();
}
```

Listing 8.18 NoteSketchActivity setCanvasImage()-Methode

```
public void setCanvasImage(Bitmap bitmap){
    if (bitmap == null){
        bitmap = Bitmap.createBitmap(mWidth, mHeight, Bitmap.Config.ARGB_8888);
        drawableImageView.setImageBitmap(bitmap, true);
    }else{
        drawableImageView.setImageBitmap(bitmap, false);
    }
    bitmap.recycle();
}
```

Erläuterung

Betrachten wir zunächst die *DrawableImageView*.

Das *Canvas* ist der Rahmen, auf dem gezeichnet wird. Die Klasse *Paint* ist für den Style und die Farben der zu zeichnenden Objekte verantwortlich. Beim Bewegen des Fingers auf dem Touch Screen erhalten wir eine Liste von Punkten, und die Klasse *Path* stellt die Methoden zur Verfügung, die Punkte zu sammeln und zu kapseln und generiert daraus eine Linie.

Der Konstruktor der *DrawableImageView* enthält den *Context* und das *AttributeSet*. Der *Context* ist wieder die Umgebung, und das *AttributeSet* enthält die Liste der Attribute des XML-Layouts. Weiter werden in der Konstruktor-Methode das *Canvas*-Objekt und das *Paint*-Objekt initialisiert. Jetzt werden *mPaint* die Eigenschaften zugewiesen.

- `mPaint.setAntiAlias(true);`: Kantenglättung
- `mPaint.setDither(true);`: Fehlerdiffusion
- `mPaint.setColor(Color.RED);`: initialer Farbwert
- `mPaint.setAlpha(45);`: der Wert, wie stark die Deckung der Linie ist
- `mPaint.setStyle(Paint.Style.STROKE);`: der Style, es wird eine Linie gezeichnet
- `mPaint.setStrokeJoin(Paint.Join.ROUND);`: die eigentliche Linienform
- `mPaint.setStrokeCap(Paint.Cap.ROUND);`: Form der Linienenden
- `mPaint.setStrokeWidth(25);`: Linienbreite

Der eigentliche Zeichenvorgang findet in `onDrawMethode()` statt. Mit `canvas.drawBitmap(mBitmap, 0, 0, null);` wird das übergebene Bitmap in das Canvas gezeichnet, und `canvas.drawPath(mPath, mPaint);` zeichnet die Linie mit den zuvor vergebenen Einstellungen. Das `mPath`-Objekt wird durch die drei Touch-Methoden mit Punkten gefüllt und wandelt diese in eine Linie um. Die `touch_start()`-Methode wird aufgerufen, wenn das Touch Display berührt wird, zunächst wird `mPath` zurückgesetzt und dann der Anfangspunkt der Linie gesetzt. Die `touch_move()`-Methode wird aufgerufen, sobald der Finger bewegt wird. Die `if`-Bedingung überprüft, ob eine Bewegung stattgefunden hat und vergleicht den absoluten Wert der Differenz von der aktuellen `x`-Koordinate und der Startkoordinate `mX` und einem willkürlich gewählten Wert der `TOUCH_TOLERANZ`. Der Vergleich wird ebenso für die Y-Koordinate durchgeführt. Ist einer der beiden Vergleiche größer oder gleich, wird mit `mPath.quadTo(...)` eine Bezier-Kurve im `mPath`-Objekt, von den Anfangskoordinaten bis zu den aktuellen Koordinaten erzeugt. Am Ende, wenn der Finger das Touch-Display verlässt, erfolgt der Aufruf der `touch_up()`-Methode. Zunächst erzeugt die Methode `mPath.lineTo(mX, mY)` die endgültige Linie, die gezeichnet werden soll, unter Zuhilfenahme der schon in `mPath`-Objekt verfügbaren Bezier-Kurve zum eigentlichen Anfangspunkt. Der Aufruf von `mCanvas.drawPath(...)` zeichnet nun die Linie `mPath` mit den festgelegten Eigenschaften von `mPaint`. Zum Schluss wird `mPath` wieder mit `mPath.reset()` in den Ursprungszustand zurückgesetzt.

Die Methoden `setPaintColor(int Color)` und `setPaintStroke(int stroke)` verändern die Farbe und die Stärke der zu zeichnenden Linie, also die Eigenschaften von `mPaint`. Der Aufruf der Methoden erfolgt bei Betätigen eines Toggle-Buttons oder der `SeekBar` in der Bedienleiste. Zunächst sendet der Listener ein Ereignis an die Activity, und dort werden die Methoden `setPaintColor()` oder `setPaintStroke()` aufgerufen und an den im jeweiligen Parameter übergebenen Wert weitergereicht. Die Funktionsweise des Interface haben wir im vorherigen Abschnitt beschrieben.

Die `setImageBitmap()`-Methode stammt aus der Klasse `ImageView`. Sie wird überschrieben und um den Parameter `bg` des Typs `boolean` erweitert. Die Methode wird in der Activity aufgerufen, sie erhält ein Bitmap-Objekt, das in der Activity erzeugt wurde, sowie `true` oder `false` als Parameter. `True` wird übergeben, wenn das Bitmap-Objekt programmäßig neu erzeugt wurde und die `mCanvas.drawColor(Color.WHITE);` ausgeführt werden muss. Dies ist notwendig, damit das Bitmap einen weißen Hintergrund erhält. Wird die Methode nicht ausgeführt und das Bitmap gespeichert, nachdem darauf gezeichnet wurde, hätte das neu erzeugte Bild einen schwarzen Hintergrund. Bei einem Bitmap, das aus einem Bild erzeugt wurde, ist dieser Aufruf nicht notwendig. Das übergebene Bitmap-Objekt kann nicht einfach dem `mBitmap` zugewiesen werden, sondern kann nur kopiert werden. Die Anweisung `mBit`

map = bm.copy(Bitmap.Config.ARGB_8888, true); kopiert das übergebene Bitmap-Objekt mit Angabe der Konfiguration, und mit der Angabe, dass dieses Bitmap veränderbar ist. Die *setImageBitmap()*-Methode muss zwingend in der Activity ausgeführt werden, da sonst das Canvas *mCanvas* nicht vollständig initialisiert ist.

Betrachten wir nun den Teil der *NoteSketchActivity*. Wir haben festgestellt, dass es unpraktisch ist, wenn während der Bearbeitung der Bildschirm gedreht wird, und sich das angezeigte Bild ebenfalls dreht. Um dies zu verhindern, wird die Ausrichtung mit der Methode *setRequestedOrientation(...)* fest auf den Portrait-Modus eingestellt. Da zur Erzeugung eines geeigneten Bitmap-Objektes die Eigenschaften der Anzeige wichtig sind, müssen diese ermittelt werden. Der Aufruf von *getWindowManager().getDefaultDisplay()* gibt die Referenz auf die Standardanzeige zurück, und diese wird in *display* gespeichert. In der nächsten Anweisung werden die Eigenschaften der Anzeige ermittelt und in der Variable *displayMetrics* festgehalten. In den weiteren Schritten werden die Höhe, die Breite und die Auflösung ermittelt und in den Variablen *mHeight*, *mWidth* und *mDensity* gespeichert. In der folgenden Anweisung *physDisplayPixel =...* erfolgt die Berechnung der realen Auflösung. Im nächsten Schritt erfolgt die Ihnen schon bekannte Referenzierung der *DrawableImageView*, und es wird die Methode *setCanvas(null)* ausgeführt. In dieser wird die zwingend notwendige Methode *.setImageBitmap(bitmap, true);* der *DrawableImageView* aufgerufen.

Die *setCanvasImage(Bitmap bitmap)*-Methode wird beim Erstellen der Activity in der *onCreateMethode()* mit **null** aufgerufen, damit ist *bitmap=null*. Die folgende *if*-Anweisung überprüft, ob *bitmap=null* ist, und trifft dies zu, wird ein *bitmap*-Objekt mit *create Bitmap(mWidth, mHeight, Bitmap.Config.ARGB_8888)* der Bitmap-Klasse erstellt. Die ersten Parameter, die das neue Bitmap-Objekt haben soll. sind Breite und Höhe. Die übergebenen Werte sind die in der *onCreateMethode()* ermittelten Display-Werte. Der nächste Parameter ist das Format, wie die Pixel später im Bitmap gespeichert werden. Die nächste Anweisung ruft die Methode *setImageBitmap()* der *DrawableImageView*-Klasse auf, übergibt das neu erstellte *Bitmap*-Objekt und *true*, und veranlasst damit, dass das *mCanvas.drawColor(Color.WHITE)* in der *Drawable* ausgeführt wird. Im folgenden *else*-Zweig wird *false* übergeben, damit die *.drawColor()*-Methode nicht ausgeführt wird, da *else* ausgeführt wird, wenn ein Bild übergeben wird.

Der Aufruf von *bitmap.recycle()* kennzeichnet das Bitmap-Objekt für den *Garbage-Collector*, das dieser das Bitmap-Objekt beim nächsten Lauf entsorgen kann.



ÜBUNG: Das Speichern von Dateien haben wir ausführlich in Kapitel 6 erläutert. Erstellen Sie eine passende Methode zum Speichern der neu erzeugten Bitmaps im Unterordner */pic*.

Listing 8.19 Bitmaps komprimieren

```
FileOutputStream fos = null;
try {
    fos = new FileOutputStream (file);
    bm.compress (CompressFormat.JPEG, 95, fos);
```

■ 8.3 Activity for Result und Grafikbearbeitung

Im vorigen Abschnitt haben wir Ihnen gezeigt, wie Sie ein Bild aus der Galerie auswählen und so umwandeln, dass dieses Bild bearbeitet werden kann. Bildbearbeitung ist immer ein sehr ressourcenhungriger Prozess. Auf einem Smartphone oder Tablet ist der Arbeitsspeicher und die Leistung der CPU begrenzt. Die Bilder der Kameraaufnahmen haben meist eine Größe von einem Megabyte und mehr. Der für die App verfügbare Speicher ist viel geringer. Außerdem sollten Sie immer bedenken, dass noch sehr viele ältere Geräte eingesetzt werden, und dass eine Vielzahl von Geräten mit nicht so üppiger Hardware wie Nexus-Geräte verwendet werden.

Um Bilder bearbeiten zu können, ist es deshalb notwendig, dass diese so decodiert bzw. skaliert werden, dass Sie zum einen noch gut erkennbar sind und zum anderen die Ressourcen im System weitgehend geschont werden. Um Bilder aus der Galerie auswählen zu können, wird ein Mechanismus benötigt, der nach dem Schließen ein Bild oder den Pfad zu der Bilddatei zurückliefert. Dies macht die *Activity for Result* möglich.

8.3.1 Mit Activity for Result den Pfad zum Bild ermitteln und sich das Ergebnis in einem Toast anzeigen lassen

In diesem Abschnitt zeigen wir Ihnen, wie Sie mit einer Activity ein Bild aus der Galerie auswählen und als Ergebnis ein URI zurück erhalten. Ein URI (*Uniform Resource Identifier*) ist ein nach einem Schema festgelegter Identifikator, der hier eine physikalische Ressource identifiziert. Im wahren Leben ist z.B. Ihre vollständige Adresse eine URI (Land + PLZ + Straße + Nr.)

1. Erstellen Sie die Methode *getImage()* in der *NoteSketchKlasse*, die einen Intent zum Aufrufen der *Activity for Result* definiert und diese startet (Listing 8.20).
2. Überschreiben Sie die Methode *onActivityResult()*, um nach Rückgabe der Daten weitere Methoden aufrufen zu können (Listing 8.21).
3. Erstellen Sie einen Menüeintrag im Note-Menü und rufen damit die Methode *getImage()* auf. Dies sollte Ihnen jetzt ohne Probleme gelingen.

Listing 8.20 Methode *getImage()* der NoteSketch-Klasse

```
public void getImage(){
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT,
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    intent.setType("image/*");
    startActivityForResult(intent, 1);
}
```

Listing 8.21 Methode *onActivityResult()* der NoteSketchActivity

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
```

```
if (requestCode == 1 && resultCode == RESULT_OK && data != null) {  
    Uri uri = data.getData();  
    if (getBitmap(uri)){  
        setCanvasImage(bitmap);  
    }  
    Log.d(TAG, "ActivityResult uri: " + uri.toString());  
}  
}
```

Erläuterung

Zunächst wird beim Betätigen des Menüeintrages die Methode `getImage()` aufgerufen. In der Methode erfolgt zuerst die Definition des Intents. Dieser Intent erhält als ersten Parameter die Absicht `Intent.ACTION_GET_CONTENT`, die dem System mitteilt, dass dieser Intent Daten zurückerhalten möchte. Im zweiten Parameter erfolgt die Festlegung, von welchem Ziel die Daten benötigt werden. Mit `intent.setType("image/*")`; definiert der Intent, welche Art von Daten abgefragt werden sollen.

Die folgende Anweisung `startActivityForResult(intent, 1);` startet eine Activity, und das System entscheidet anhand des gesendeten Intents, welche Anwendung dies ist. Sind mehrere vorhanden, so erscheint ein Chooser, der die möglichen Apps auflistet. Der zweite Parameter ist eine Id, um das Ergebnis identifizieren zu können.

Die `onActivityResult()`-Methode wird ausgeführt, wenn die aufgerufene Activity ein Ergebnis sendet und erhält als Parameter den Wert von `requestCode`, `resultCode` und die vom Intent angefragten Daten. Die if-Anweisung überprüft, ob die drei Bedingungen `requestCode == 1`, `resultCode == RESULT_OK`, `data != null` erfüllt sind. Nur wenn alle drei Bedingungen erfüllt sind, werden die Anweisungen ausgeführt. Trifft nur eine Bedingung nicht zu, wird die Methode verlassen.

8.3.2 Ein Bild in ein Bitmap umwandeln

Im letzten Abschnitt haben wir gezeigt, wie die URI zu einem Bild ermittelt wird.

Jetzt erstellen Sie

1. die Methode `getBitmap()` in der `NoteSketchActivity` (Listing 8.22),
2. in der `FileHelper`-Klasse, die Methode `getPathFromUri()` (Listing 8.23) und
3. ebenfalls in der `NoteSketchActivity` die Methode `calculateSampleSize()` (Listing 8.24).

Listing 8.22 Methode `getBitmap()` der `NoteSketchActivity`

```
private boolean getBitmap(Uri uri) {  
    boolean result = false;  
    int reqHeight = mHeight;  
    int reqWidth = mWidth;  
    try {  
        String bitmapPath = fileHelper.getPathFromUri(getApplicationContext(), (uri));  
        ExifInterface exifInterface = new ExifInterface(bitmapPath);  
        int exifWidth = Integer.parseInt(exifInterface.getAttribute  
            (ExifInterface.TAG_IMAGE_WIDTH));  
    }  
}
```

```

        int exifHeight = Integer.parseInt(exifInterface.getAttribute
                (ExifInterface.TAG_IMAGE_LENGTH));
        if(exifHeight < exifWidth){
            reqHeight = mWidth;
            reqWidth = mHeight;
        }
        BitmapFactory.Options bfOptions = new BitmapFactory.Options();
        bfOptions.inPreferredConfig = Bitmap.Config.ARGB_8888;
        bfOptions.inDither = true;
        bfOptions.inMutable = true;
bfOptions.inJustDecodeBounds = true;
        Bitmap bfBitmap = BitmapFactory.decodeFile(bitmapPath, bfOptions);
        int sampleSize = calculateInSampleSize(bfOptions, reqHeight, reqWidth);
        bfOptions.inTempStorage = new byte[16 * 1024];
        bfOptions.inSampleSize = sampleSize;
bfOptions.inJustDecodeBounds = false;
        bfBitmap = BitmapFactory.decodeFile(bitmapPath, bfOptions);
        bfBitmap = Bitmap.createScaledBitmap(bfBitmap, reqWidth, reqHeight, true);
        Matrix matrix = new Matrix(drawableImageView.getMatrix());
        if (bfBitmap.getHeight() < bfBitmap.getWidth())
        {
            matrix.preRotate(90);
        }
        bitmap = Bitmap.createBitmap(bfBitmap, 0, 0, reqWidth, reqHeight, matrix,
        true); bfBitmap.recycle();
        bfBitmap = null;
        result = true;
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return result;
}

```

Listing 8.23 Methode getPathFromUri() der Klasse FileHelper

```

public String getPathFromUri(Context context, Uri uri) {
    String[] projection = { MediaColumns.DATA };
    CursorLoader cursorLoader = new CursorLoader(context, uri, projection, null,
    null, null);
    Cursor cursor = cursorLoader.loadInBackground();
    cursor.moveToFirst();
    int column_index = cursor.getColumnIndexOrThrow
            (MediaStore.Images.Media.DATA);
    String result = cursor.getString(column_index);
    cursor.close();
    return result;
}

```

Listing 8.24 Methode calculateSampleSize() Klasse NoteSketchActivity

```

public static int calculateInSampleSize(BitmapFactory.Options options, int reqHeight,
int reqWidth) {
    // Raw height and width of image
    final int height = options.outHeight;
    final int width = options.outWidth;
    int inSampleSize = 1;
    if (height > reqHeight || width > reqWidth) {

```

```
    final int heightRatio = Math.round((float) height / (float) reqHeight);
    final int widthRatio = Math.round((float) width /
                                    (float) reqWidth);
    inSampleSize = heightRatio < widthRatio ? heightRatio : widthRatio;
}
return inSampleSize;
}
```

Erklärung

Der Aufruf der `getBitmap()`-Methode erfolgt in der zweiten `if()`-Anweisung der Methode `onActivityResult()` und erhält als Parameter die URI des gewählten Images. Als Erstes wird die Ergebnisvariable `result` deklariert und erhält den Wert **false**. Dann erfolgt die Deklaration der beiden Variablen für die Zielgröße des Bildes, und diesen werden die ermittelten Display-Werte zugewiesen.

Im `try`-Block erfolgt als Erstes die Deklaration der String-Variablen für den absoluten Pfad im Dateisystem des Bildes und wird sofort mit der `FileHelper`-Methode `getPathFromUri()` initialisiert. Die Methode `getPathFromUri()` enthält als Parameter die erhaltene URI und liefert den Dateipfad als String zurück. Im nächsten Schritt wird das `ExifInterface` mit dem Pfad zum Image initialisiert. Mit dem `ExifInterface` können im Bild gespeicherte Metadaten ausgelesen werden. Diese Metadaten sind neben der Breite und Länge auch Kamerahersteller, Aufnahmedatum etc. Die nächsten beiden Anweisungen ermitteln die Breite `exifWidth` und Höhe `exifHeight` des Image als Integer. Mit `Integer.parseInt()` kann eine als String vorliegende Zahl in den Integer-Typ umgewandelt werden.

In der folgenden `if`-Anweisung `if(exifHeight < exifWidth)` werden die Image-Höhe und -Breite verglichen, ist die Höhe des Bildes kleiner als die Breite, werden Zielbreite und -höhe vertauscht.

Ein Bild wird in einer festen Ausrichtung von der Kamera gespeichert. Wird die Kamera beim Fotografieren um 90° gedreht, so ist die Breite größer als die Höhe. Damit das Bild in den richtigen Maßen nach der Decodierung vorliegt, müssen Zielbreite und -höhe angepasst werden. Es wären sonst Teile des Bildes abgeschnitten.

Die Decodierung des Image erfolgt durch die Klasse `BitmapFactory`. Diese Klasse bietet Methoden zur Erstellung von Bitmap-Objekten aus unterschiedlichen Quellen an. Diese Quellen können Dateien, Streams und Byte-Arrays sein. Über verschiedene Optionen kann die Decodierung konfiguriert werden. Zur Konfiguration der Optionen wird die innere Klasse der `BitmapFactory()`-Optionen verwendet.

Mit der Anweisung `BitmapFactory.Options bfOptions = new BitmapFactory.Options();` erfolgt die Initialisierung der Optionen. In den folgenden Anweisungen werden die Optionen parametrisiert. Als Erstes erfolgt die Zuweisung des präferierten Bildformats mit `.inPreferredConfig`. Mit `.inDither` wird das Dithering eingeschaltet. Der nächste Parameter `inTempStorage` legt fest, dass ein 16-kByte-großer Puffer im Temp Storage genutzt werden soll, und `inMutable= true` legt fest, dass das erstellte Bitmap veränderlich ist. Die folgende Option `bfOptions.inJustDecodeBounds = true;` ist von besonderer Bedeutung. Ihnen wird sicherlich nicht entgangen sein, dass die Methode `BitmapFactory.decodeFile()` zweimal ausgeführt wird. Beim erstmaligen Aufruf der Methode ist die Option `.inJustDecodeBounds = true` und bewirkt, dass das Image in einer Art Analysemodus decodiert wird. Es

wird kein Bitmap-Objekt *bitmap* erstellt. Dies erfolgt, um Informationen zu ermitteln, die für die Berechnung der Samplesize wichtig sind. Die Samplesize ist der Faktor, um den das Image vergrößert wird. Diesen Vorgang könnten wir uns sparen, da wir diese Informationen schon aus den Exif-Informationen am Anfang der Methode gewonnen haben. Im Moment gehen wir noch davon aus, dass in der Galerie nur Bilder auswählbar sind, die mit der Kamera erzeugt wurden, die diese Informationen bereitstellen.

Nachdem nun die *BitmapFactory.decodeFile()* mit den Parametern Dateipfad des Image und den Optionen das erste Mal ausgeführt wurde, folgt die Anweisung `int sampleSize = calculateInSampleSize(bfOptions, reqHeight, reqWidth);`. Hier wird die Variable *sampleSize* mit dem berechneten Wert aus der Methode *calculateInSampleSize()* deklariert. Die Methode liefert den kleinsten Wert zurück, der in der *if*-Anweisung der Methode errechnet wurde. Dieser Wert wird jetzt mit *.inSampleSize* in den Optionen festgelegt. Durch *.inJustDecodeBounds=false* erfolgt im nächsten Aufruf der Methode die Erstellung des Bitmaps *bfBitmap* durch mit dem Wert *sampleSize* erweiterte Optionen. Durch die folgende Initialisierung der Klasse Matrix erfolgt die direkte Zuweisung des Wertes der Matrix von *drawableImageView*. Die Matrix wird benötigt, um das Bitmap-Objekt später transformieren zu können. Die nächste *if*-Anweisung prüft, ob die Höhe des erzeugten Bitmaps größer die Breite ist. Trifft dies zu, ist die Bedingung erfüllt, und die Matrix wird um 90° gedreht. Im folgenden Schritt erhält das Bitmap-Objekt *bitmap* ein neues Bitmap, das durch die Methode *.createBitmap* aus dem decodierten Bitmap *bfBitmap* erstellt wird. Die weiteren Parameter der Methode sind die Startwerte der ersten Pixel, die zu verwendende Matrix und *true* für den Filter. Im Anschluss werden *bfBitmap.recycle()*, *bitmap=null* und der Rückgabewert *result=true* gesetzt. Am Ende der Methode wird *result* zurückgegeben. Da *result* am Anfang mit dem Wert *false* initialisiert wurde, ist es nicht nötig, im *catch*-Zweig *result=false* aufzurufen.

Zum Schluss dieses Abschnittes betrachten wir noch kurz die Methode *getPathFromUri()*. Zunächst wird das String-Array *projection* erzeugt, das aus dem Interface *MediaStore.MediaColumns* das Schema für DATA erhält. Das Schema enthält alle Informationen zu Dateien. Der nächste Schritt erzeugt einen *CursorLoader*, der sehr abstrakt eine Abfrage formuliert und als Parameter übergibt. Die übergebenen Parameter sind der *Context*, URI und *projection*. Die weiteren Parameter *selection*, *selectionArgs* und *sortOrder* erhalten *null*. Content Provider sind nach einem festen Schema aufgebaut, und somit kann auf diese Weise eine Query formuliert werden. Die hier formulierte Query, es sollen alle Datei-Informationen zu der URI des Content-Providers *MediaStore.MediaColumns* geladen werden. Der *CursorLoader* lädt die Daten, und der Cursor durchläuft diese zeilenweise. Jedoch beginnt der Cursor, sobald erste Daten verfügbar sind, und nicht erst, wenn die gesamten Daten der Abfrage zur Verfügung stehen. Weiterhin wird das Laden der Daten beendet, sobald der Cursor geschlossen wird.

Im nächsten Schritt wird der *cursor* durch *cursor.loadInBackground()* initialisiert, und die Daten werden vom *ContentProvider* schon abgefragt. Ein Cursor ist ein Zeiger, der innerhalb einer zeilenbasierten Ergebnisliste navigiert werden kann.

Mit *cursor.moveToFirst();* wird der Cursor in die erste Zeile der Datenliste positioniert. Mit der folgenden Anweisung wird der Spaltenindex ermittelt und der Integer-Variablen *column_index* zugewiesen. Die nächste Anweisung erzeugt den String *result*, dem durch *cursor.getString(column_index);* der Dateipfad der URI zugewiesen wird. Jetzt wird der Cursor

durch die `.close()`-Methode geschlossen, und der Result zurück an die aufrufende Methode übergeben.



HINWEIS: Ein Cursor muss am Ende immer geschlossen werden.

■ 8.4 Zwischenstand der App (Version 0.5)

In diesem Kapitel haben wir der Anwendung die Funktionalität zur Erstellung einer Bild-Notiz hinzugefügt. Der User besitzt nun die Möglichkeit, beliebige Bilder aus der Galerie auszuwählen und diese dann mit Markierungen zu versehen, wie in Bild 8.3 zu sehen. Außerdem können direkt auf einer weißen Leinwand Skizzen erstellt werden und die Leinwand oder das Bild immer wieder neu geladen werden.

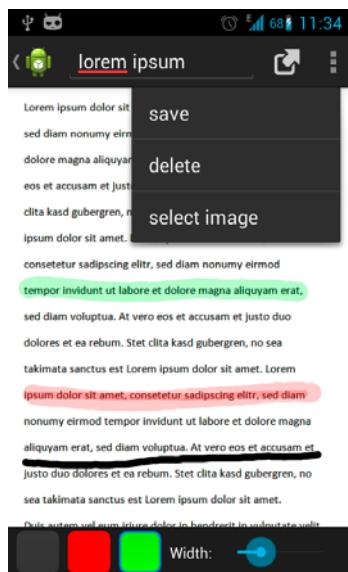


Bild 8.3

Notiz mit importiertem Text-Bild und Markierung

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 wiedergegeben 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.

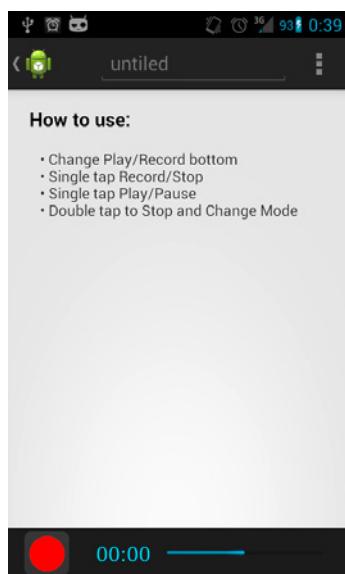


Bild 9.1
Ansicht NoteVoiceActivity

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.set AudioSource(MediaRecorder.AudioSource.MIC);
    mRecorder.set OutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    mRecorder.set AudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
    mRecorder.set MaxDuration(3598000);
    mRecorder.set OutputFile(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.

10

Dialog-Fragments und Datenbank

In Kapitel 9 haben wir die Anwendung um die Möglichkeiten zum Aufnehmen, Abspielen und Speichern von Sprachnotizen erweitert, wir haben die Nutzung von einfachen Gesten gezeigt und Threads erläutert.



Die Beispieldateien zu diesem Kapitel finden Sie unter
www.downloads.hanser.de im Unterordner *scytenotes 0.7*.

Nachdem die App nun bereits über solide Grundfunktionalitäten für die Erstellung und Verwaltung von Notizen verfügt, werden wird die App in diesem und den folgenden Kapiteln um zeit- und ortsbasierte Erinnerungen ergänzen. Die dafür notwendigen Informationen werden nicht im Dateisystem, sondern in einer Datenbank verwaltet. Dieses Kapitel betrachtet folgende Themen:

- Dialog-Fragments für Auswahl- und Eingabedialoge
- Datenbank erstellen
- Daten in die Datenbank einfügen, ändern und löschen
- Daten in der ListView anzeigen

■ 10.1 Dialog-Fragments

In Kapitel 8 wurden die Fragments bereits erläutert. Mit dem Api-Level 11 wurden Dialog-Fragments eingeführt. Mit den Dialog-Fragments hat der Entwickler nun die Möglichkeit, Dialoge einfach und schnell anzupassen und einen Container für komplexe Dialoge zu erstellen. Bis zur Einführung von Dialog-Fragments war es sehr aufwändig, Dialoge zu erstellen, die vorhandene Dialoge erweitern oder kombinieren.

In der Beispiel-App erstellen wir zunächst zwei Dialoge, die beim erstmaligen Speichern einer Textnotiz aufgerufen werden sollen.

Der *ReminderSelection-Dialog* (Bild 10.1) wird nach dem Speichern durch die Methode *showSelectReminderDialog()* aufgerufen. In diesem Dialog wählt der User den Typ der Erinne-

rung aus. Wurde *none* gewählt, schließt sich der Dialog, wurde *time* gewählt, erscheint der *DateTime*-Dialog.

Der *ReminderSelection*-Dialog ist vollständig im Code der Klasse *ReminderSelectionDialogFragment* (Listing 10.1) definiert.

Listing 10.1 ReminderSelection-Klasse, Auswahldialog der Erinnerungen

```
public class ReminderSelectionDialogFragment extends DialogFragment {
    ReminderType reminderType;
    public static ReminderSelectionDialogFragment newInstance() {
        ReminderSelectionDialogFragment reminderSelectionDialogFragment
            = new ReminderSelectionDialogFragment();
        return reminderSelectionDialogFragment;
    }
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder =
            new AlertDialog.Builder(getActivity());
        builder.setTitle(R.string.select_reminder)
            .setItems(R.array.reminder_name,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int which) {
                        Log.d("ReminderDialog", String.valueOf(which));
                        switch (which){
                            case 0 :
                                reminderType = ReminderType.none;
                                break;
                            case 1:
                                reminderType = ReminderType.time;
                                break;
                            case 2:
                                reminderType = ReminderType.location;
                                break;
                            default: break;
                        }
                    rSDListener.
                        onReminderSelectionItemClick(
                            ReminderSelectionDialogFragment.this
                            , reminderType);
                }
            });
        return builder.create();
    }
    public interface ReminderSelectionDialogListener{
        public void onReminderSelectionItemClick(
            ReminderSelectionDialogFragment dialog
            ,ReminderType reminderType);
    }
    ReminderSelectionDialogListener rSDListener;

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            rSDListener = (ReminderSelectionDialogListener) activity;
        } catch (ClassCastException e) {
```

```
        throw new ClassCastException(activity.toString()
            + " must implement ReminderSelectionDialogListener");
    }
}
```

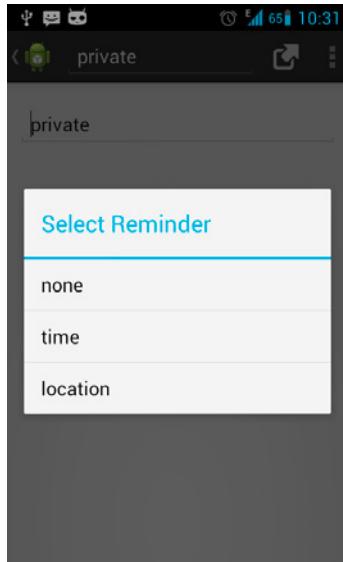


Bild 10.1
Ansicht des Dialogs Select Reminder

Listing 10.1 zeigt die Erstellung des Dialogs. Als Erstes wird eine statische Instanz des Dialog-Fragments erzeugt. Die Erstellung des Dialogs erfolgt dann in der *onCreate()*-Methode, welche die vom *AlertDialog.Builder* erstellte View zurückliefert. Als Erstes wird der *builder* der Klasse *AlertDialog.Builder* erstellt und initialisiert. Mit *builder.setTitle()* erhält der Dialog seine Überschrift. Der übergebene Parameter ist wieder eine Referenz auf einen String in den String-Ressourcen *strings.xml*. Der *AlertDialog* bietet im Standard die Nutzung einer *ListView* ohne diese explizit erstellen zu müssen. Durch die Methode *.setItems()* wird die *ListView* initialisiert und erhält als ersten Parameter die Liste der anzugezeigenden Objekte. Dies ist ein einfaches String-Array, definiert in den String-Ressourcen. Das hier genutzte Verhalten, die Erstellung eines Listadapters, ist bei einfachen String-Arrays, die Sie schon aus Abschnitt 6.3 kennen, nicht notwendig. Der zweite Parameter ist der Ereignishandler, der durch *new DialogInterface.OnClickListener()* direkt erstellt wird. Der *OnClickListener* wird benötigt, um den Klick auf ein Item in der *ListView* entgegenzunehmen und in der *onClick()*-Methode auszuwerten. Wählt der User ein Item aus, folgt die Ausführung der *onClick()*-Methode, und diese erhält den Index des gewählten Items als Parameter. In der *Switch/Case*-Anweisung erfolgt die Auswertung, und in dem Index entsprechenden Case-Zweig wird nun der Typ der gewählten Reminders festgelegt. Am Ende der *onClick()*-Methode wird der Listener vom Interface *ReminderSelectionDialogListener* aufgerufen und erhält den Reminder-Typ als Parameter übergeben.

Der Dialog wird durch die Methode *showSelectReminderDialog()* in der *NoteTextActivity* aufgerufen. Zunächst wird eine neue Instanz des Dialog-Fragments erstellt, die die statische

Instanz des Dialog-Fragments referenziert und mit der `.show()` anzeigt. Als Parameter erhält die `.show()`-Methode eine Referenz auf den Fragment-Manager der Activity, `getFragmentManager()` ermittelt die richtige Instanz des Fragment-Managers, der der Activity zugewiesen ist. Der zweite Parameter des Typs String ist der Tag, um das Dialog-Fragment identifizieren zu können.

Listing 10.2 Methode `showSelectReminderDialog()` der Klasse NoteTextActivity

```
public void showSelectReminderDialog() {
    DialogFragment reminderDialogFragment =
        ReminderSelectionDialogFragment.newInstance();
    reminderDialogFragment.show(getFragmentManager(),
        "ReminderDialog");
}
```

Die folgende Methode `onReminderSelectionItemClick()` (Listing 10.3) ist die überschriebene Methode vom Interface `ReminderSelectionDialogListener` des Dialog-Fragments. Die Funktionsweise von Interfaces und Fragments haben wir ausführlich in Abschnitt 8.1.1.1, „Daten-austausch zwischen Fragment und Activity“, erläutert.

Wählt der User im `ReminderSelection`-Dialog ein Item aus, wird die Methode `onReminderSelectionItem()` in der NoteSketchActivity aufgerufen und der Dialog geschlossen.

Die Methode erhält den gewählten Erinnerungstyp als Parameter übergeben, und in der Switch/Case-Anweisung erfolgt die Ausführung der Methoden im entsprechenden Case-Zweig. Derzeit erfolgt nur die Ausführung der Methode `showDateTimeDialog()`, um den `DateTime`-Dialog anzulegen (Bild 10.2).

Listing 10.3 Überschriebene Interface-Methode des Dialog-Fragments in der NoteSketchActivity

```
@Override
public void onReminderSelectionItemClick(
    ReminderSelectionDialogFragment rSDialog,
    ReminderType reminderType) {
    switch (reminderType) {
        case time:
            showDateTimeDialog();
            break;
        case location:
            break;
        default: break;
    }
}
```

Das Layout des `DateTime`-Dialogs `fragment_datetime_dialog.xml` ist sehr einfach. Es besteht aus einem Linear-Layout mit senkrechter Orientierung und verfügt über vier TextView-Elemente.

Lediglich die TextView-Elemente zur Anzeige von Datum und Uhrzeit erhalten zusätzlich das Style-Attribut `style="@android:style/Widget.DeviceDefault.Light.Spinner"`. Das Style-Attribut bewirkt, dass die TextViews mit einem gefüllten Dreieck in der rechten unteren Ecke dargestellt werden.

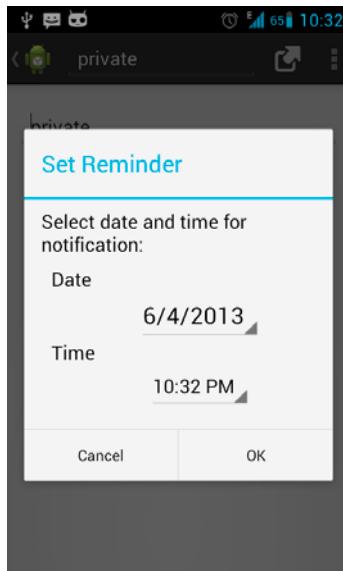


Bild 10.2
Ansicht DateTime-Dialog

Die Erstellung des Dialogs erfolgt wieder in der *onCreateDialog()*-Methode der Klasse *Date TimeDialogFragment*.

Listing 10.4 Methode *onCreateDialog()* der Klasse *DateTimeDialogFragment*

```
@Override  
public Dialog onCreateDialog(Bundle savedInstanceState) {  
    calendar = Calendar.getInstance();  
    year = calendar.get(Calendar.YEAR);  
    month = calendar.get(Calendar.MONTH);  
    day = calendar.get(Calendar.DAY_OF_MONTH);  
    hour = calendar.get(Calendar.HOUR_OF_DAY);  
    min = calendar.get(Calendar.MINUTE);  
  
    AlertDialog.Builder builder =  
        new AlertDialog.Builder(getActivity());  
    LayoutInflater inflater = getActivity().getLayoutInflater();  
    View view = inflater.inflate  
        (R.layout.fragment_datetime_dialog, null);  
    textViewDateDialog = (TextView) view  
        .findViewById(R.id.textView_date_dialog);  
    textViewTimeDialog = (TextView) view  
        .findViewById(R.id.textView_time_dialog);  
    textViewDateDialog.setOnClickListener(new OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            DatePickerDialog dateDialog = new DatePickerDialog  
(getDialog().getContext(), dateListener, year, month, day);  
            dateDialog.onCreatePanelView(STYLE_NORMAL);  
            dateDialog.show();  
        }  
    }); //Ende setOnClickListener  
    textViewTimeDialog.setOnClickListener(new OnClickListener() {  
        @Override
```

```

        public void onClick(View v) {
            TimePickerDialog timePickerDialog = new TimePickerDialog(
                getDialog().getContext(), timeListener, hour, min,
                DateFormat.is24HourFormat
                    (getActivity().getApplicationContext()));
            timePickerDialog.show();
        }
    };//Ende setOnClickListener
    updateTextviewDate();
    updateTextviewTime();
    builder.setView(view)
        .setTitle(getResources().getString(R.string.setReminder))
        .setMessage(getResources().getString(R.string.select_date_time))
        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dateTimeDialog, int id) {
                dtlListener.onDialogPositiveClick(
                    DateTimeDialogFragment.this, calendar);
            }
        });//Ende setPositiveButton
        .setNegativeButton("Cancel",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dateTimeDialog, int id) {
                    dtlListener.onDialogNegativeClick(
                        DateTimeDialogFragment.this);
                }
            });
    }

    return builder.create();
}

```

In der *onCreateDialog()*-Methode wird zunächst eine Calendar-Instanz erzeugt. Die Klasse *Calendar* ermöglicht, ein Datum in seine Komponenten Monat, Tag, Stunde des Tages und Sekunde zu zerlegen und einzeln zu setzen. Die einzelnen Werte der Komponenten von *Calendar* sind vom Typ Integer und werden den Integer-Variablen übergeben. Wir benötigen die Komponenten später bei der Verwendung des *DatePicker*- und *TimePicker*-Elements.

Im nächsten Schritt erfolgt wieder die Erstellung des *AlertDialogBuilders*, danach erfolgt die Initialisierung des *LayoutInflater*, in der nächsten Anweisung erfolgt die Erstellung einer View, die durch den Inflater mit dem Layout des Dialogs aufgefüllt wird. Der zweite Parameter ist null, da das Layout nicht in einem übergeordneten Layout erstellt wird, sondern dem Dialog später zugewiesen wird. Die nächsten beiden Anweisungen *findViewById(..)* referenzieren die beiden *TextView*-Elemente des Layouts für die Anzeige von Datum und Uhrzeit. Jedes *TextView*-Element registriert nun einen *OnClickListener*, damit beim Klicken auf das View-Element der *DatePicker*- oder *TimePicker*-Dialog aufgerufen wird. In der *onClick(View v)*-Methode wird der *DatePicker*- oder *TimePicker*-Dialog erstellt und erhält als Parameter den Kontext des Dialogs des Fragments, den Listener, der die im Dialog gewählten Werte entgegennimmt und die Werte der einzelnen Dialogkomponenten, die initial angezeigt werden. Beide Dialoge erwarten diese vom Typ Integer. Da der *Date Picker*-Dialog auch als Kalender dargestellt werden kann, wird mit der Anweisung *.onCreatePanelView(STYLE_NORMAL)* der übliche Style festgelegt. Mit *.show()* erfolgt die Anzeige des Dialogs.

Listing 10.5 zeigt den Listener *dateListener* für den *DatePicker*-Dialog. Beide Listener, der *DatePicker* oder *Time Picker*, sind gleich aufgebaut. Sie erhalten als Parameter die gewählten Werte, sodass diese den entsprechenden Integer-Variablen übergeben werden. Danach wird dem Calendar-Objekt jeder Wert zugewiesen, und über die Hilfsmethode *updateTextView...* in der Methode für die jeweilige TextView *Date* oder *Time* wird das Datum oder die Uhrzeit als String aus dem Calendar-Objekt gebildet und der TextView übergeben.

Die weiteren Schritte der *onCreateDialog()*-Methode entsprechen fast der des *Reminder Selection*-Dialogs. Der Aufruf von *builder.setView()* registriert nun die durch den LayoutInflater erstellte View, und mit *.setMessage()* wird ein Text aus den String-Ressourcen angezeigt.

Listing 10.5 Interface des DatePicker-Dialogs

```
private DatePickerDialog.OnDateSetListener dateListener =
    new DatePickerDialog.OnDateSetListener() {
    @Override
    public void onDateSet(DatePicker view,
        int yr, int monthOfYear, int dayOfMonth) {
        year = yr;
        month = monthOfYear;
        day = dayOfMonth;
        calendar.set(year, month, day);
        updateTextViewDate();
    }
};
```

Die Klasse *DateTimeDialogFragment* implementiert ebenfalls ein Interface, das ein Calendar-Objekt an die Methode *onDialogPositiveClick()* der NoteTextActivity übergibt.

Listing 10.6 Interface des DateTime-Dialog-Fragments mit Listener

```
public interface DateTimeDialogListener {
    public void onDialogPositiveClick(DialogFragment dateTimeDialog,
        Calendar calendar);

    public void onDialogNegativeClick(DialogFragment dateTimeDialog);
}
//Listener des Interface
DateTimeDialogListener dtlListener;
```

10.1.1 Klasse für Erinnerungen, Reminder

Damit die Verwaltung der einzelnen Erinnerungen möglichst einfach ist, wird die Klasse *Reminder* benötigt. Die Klasse *Reminder* dient später auch als Datenmodell der Datenbank. Außerdem organisieren Sie jetzt den Sourcecode in Pakete, um die Übersicht der einzelnen Klassen zu verbessern.

1. Erstellen Sie als Erstes ein neues Unterpaket *data* vom Programm paket *eu.scyte.notes*.
2. Verschieben Sie alle Klassen, die nicht Activities, Fragmente oder Items sind, also alles, was nicht mit der Oberfläche in Verbindung steht, in das neue Paket *eu.scyte.notes.data*. Führen Sie diesen Vorgang unbedingt in Eclipse mit Drag & Drop durch. Eclipse refrak-

turiert den Code und aktualisiert automatisch alle Verweise, sodass Sie keine Änderungen im Code selbst durchführen müssen (Bild 10.3).

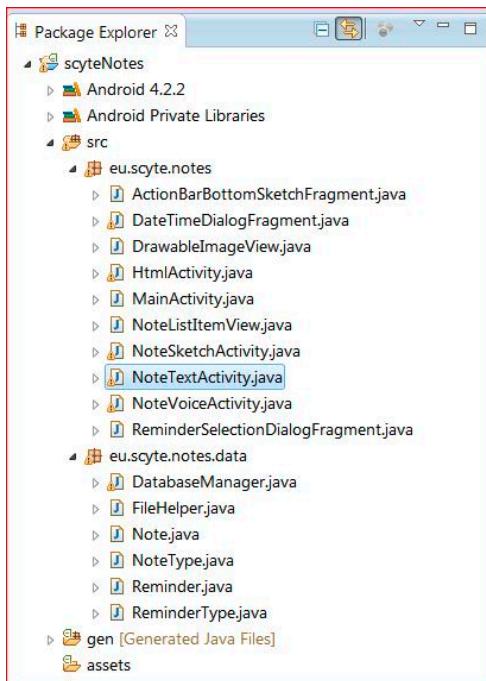


Bild 10.3
Ansicht Package Explorer in Eclipse

3. Erstellen Sie nun die Klasse *Reminder* im Paket *eu.scyte.notes.data* (Listing 10.7).

Listing 10.7 Ausschnitt der Klasse Reminder

```
public class Reminder {

    private long m DataBaseId = -1;
    private String m FilePath;
    private String m Subject;
    private NoteType m NoteType;
    private ReminderType m ReminderType;
    private String m Date Time;
    private double m Latitude;
    private double m Longitude;
    private String m Address;
    private String m SubAddress;

    public Reminder(String filePath
                    , String subject
                    , NoteType noteType) {
        m FilePath = filePath;
        m Subject = subject;
        m NoteType = noteType;
    }
}
```

```
public String getDateTime() {
    return mDateTime;
}
public void setDateTime(Long dateTimeInMilliSecond) {
    if(dateTimeInMilliSecond != null
        && dateTimeInMilliSecond > 0){
        mReminderType = ReminderType.time;
        mDateTime = dateTimeInMilliSecond;
    }
}
...
```

Die Verwendung der *Reminder*-Klasse erfolgt in der Methode *onDialogPositiveClick()* der NoteTextActivity des Interface *DateTimeDialogListener* (Listing 10.8).

Listing 10.8 Interface-Methode *onDialogPositiveClick()* der NoteTextActivity

```
@Override
public void onDialogPositiveClick(DialogFragment dialog,
                                  Calendar calendar) {
    Reminder dateTimeReminder =
        new Reminder(oldFile.getAbsolutePath(),
                    fileHelper.deleteFileExtension(oldFile.getName()),
                    NoteType.text);
    dateTimeReminder.setDateTime(calendar.getTimeInMillis());
}
```

Die Methode wird vom Listener *DateTimeDialogListener* des *DateTimeDialog*-Fragments aufgerufen und erhält neben dem Dialog ein *Calendar*-Objekt als Parameter.

Bei der Initialisierung des *dateTimeReminder*-Objekts durch Aufruf des Konstruktors der *Reminder*-Klasse müssen als Parameter der absolute Dateipfad als String, der Name ohne Dateierweiterung und der Typ der Notiz übergeben werden. Da die Ausführung in der Text-Activity stattfindet, kann der Typ der Notiz direkt ohne Prüfung festgelegt werden. Die *Reminder*-Methode *.setDateTime()* erwartet einen Wert des Typs *long*. Aus dem erhaltenen *Calendar*-Objekt kann die Zeit mit der Methode *getTimeInMillis()* abgerufen werden. So kann dem *Reminder*-Objekt die Zeit ganz einfach übergeben werden.

Sie werden sich sicherlich fragen, weshalb die Zeit weiter im Datentyp *Long* festgehalten wird. Da es nötig ist, die Zeit bzw. Datum und Zeit in verschiedenen Ausgabeformaten darzustellen und sich diese nur durch Ändern der Spracheinstellungen des Telefons verändern können, ist die Nutzung von *Long* recht einfach. Mit einfachen Methoden ist nun eine sprachabhängige Darstellung zur Laufzeit möglich. Dies haben Sie schon in der Klasse *ListView* bei der Darstellung des Dateidatums gelernt.

■ 10.2 Datenbank in Android verwenden

Auch auf einer mobilen Plattform darf ein Datenbankmanagementsystem nicht fehlen, um strukturiert Daten zu speichern und zu organisieren. Auf einem mobilen System werden jedoch andere Ansprüche an das Datenbanksystem gestellt als in Web- oder Desktop-

Anwendungen. Die Android-Entwickler haben sich sehr frühzeitig bei der Entwicklung von Android für SQLite¹ als transaktionales Datenbanksystem entschieden und unterstützen dies seit der ersten Version. SQLite ist eine performante und robuste Embedded-Datenbank. Android stellt über mehrere einfache Klassen den Zugriff zu SQLite zur Verfügung. Die Anforderungen, wie geringer Speicherverbrauch, null Konfiguration und einfache Programmierschnittstelle, erfüllt diese kompakte (wenige 100 kB) Datenbank-Bibliothek außerordentlich gut. SQLite existiert seit dem Jahr 2000, ist komplett in C geschrieben und wird auf den unterschiedlichsten Plattformen unter anderem in Browsern, Bildbearbeitungssoftware oder Mail-Anwendungen eingesetzt. In den folgenden Abschnitten erhalten Sie einen einfachen Einstieg in die Datenbankwelt.

In Android haben Sie verschiedene Möglichkeiten, Datenbankabfragen (DML) bzw. Datendefinitionen (DDL) auszuführen. Neben puren SQL-Abfragen, die mit `.execSQL` ausgeführt werden, existieren noch die Cursor-Query und die Prepared-Statements. Aus Gründen der Performance sind Prepared-Statements oder Cursor-Queries zu bevorzugen.

Zum Speichern der Daten bietet SQLite drei Datentypen an:

- INTEGER
- REAL
- TEXT

10.2.1 Datenbank erstellen

Eine SQLite-Datenbank ist schnell erstellt. Bevor wir nun aber die Datenbank erstellen, müssen wir das Datenmodell festlegen. Weiterhin erstellen wir für den Zugriff auf die Datenbank eine eigene Hilfsklasse, die alle Methoden zum Speichern, Ändern und Löschen der Einträge bereitstellt.

Die Klasse `DatabaseManager` ist die Hilfsklasse, um die Erinnerungen in der Datenbank zu verwalten. Die Klasse erbt vom `SQLiteOpenHelper`, und somit stehen die Methoden zum Erstellen von Datenbanken und Tabellen zur Verfügung. In dieser Klasse werden zunächst der Name der Datenbank, die Version, der Tabellenname und die Spalten der Tabelle als statische String-Variablen definiert (Listing 10.9).

```
Listing 10.9 Signatur und statische Variablen der Klasse DatabaseManager
public class DatabaseManager extends SQLiteOpenHelper {

    /** DATABASE */
    private final static String DATABASE_NAME = "notes_reminder.db";
    private final static int DATABASE_VERSION = 1;

    /** TABLE */
    private final static String REMINDER_TABLE = "reminder";

    /** COLUMNS */
    private final static String TABLE_ID = "_id";
    private final static String FILE_PATH = "filePath";
```

¹ www.sqlite.org

```

private final static String SUBJECT = "subject";
private final static String CATEGORY = "category";
private final static String TYPE = "type";
private final static String DATE_TIME = "date_time";
private final static String LATITUDE = "latitude";
private final static String LONGITUDE = "longitude";
private final static String ADDRESS = "address";
private final static String SUB_ADDRESS = "sub_address";
private final static String IS_NEW = "is_new";
private final static String IS_NEAR = "is_near";

```

Listing 10.10 Methode onCreate() der Klasse DatabaseManager

```

public void onCreate(SQLiteDatabase db) {
String createTABLE = "CREATE TABLE " + REMINDER_TABLE + " (" +
        + TABLE_ID + " INTEGER PRIMARY KEY autoincrement," +
        + FILE_PATH + " TEXT NOT NULL," +
        + SUBJECT + " TEXT," +
        + CATEGORY + " TEXT," +
        + TYPE + " TEXT," +
        + DATE_TIME + " DATETIME," +
        + LATITUDE + " REAL," +
        + LONGITUDE + " REAL," +
        + ADDRESS + " TEXT," +
        + SUB_ADDRESS + " TEXT," +
        + IS_NEW + " INTEGER," +
        + IS_NEAR + " INTEGER," +
        + " CONSTRAINT " +
        + FILE_PATH +
        + " UNIQUE (" + FILE_PATH + " )"
        ON CONFLICT ROLLBACK
    +
);
    db.execSQL(createTABLE);
}

```

Der bei der Anlage der Klasse automatisch erstellte Konstruktor *public DatabaseManager* wurde angepasst, sodass nur der Kontext als Parameter übergeben werden muss. Der Datenbankname und die Version sind durch die statischen Variablen *DATABASE_NAME* und *DATABASE_VERSION* schon definiert. Der Parameter für die Cursor Factory erhält den Wert null, und somit wird automatisch die Standard-Cursor Factory verwendet.

Der Aufruf der *onCreate()*-Methode erfolgt immer dann, wenn noch keine Datenbank existiert. Der String *createTable* enthält den vollständigen SQL-Befehl zur Erstellung der Tabelle, ein Aufruf von *create Database*, wie Sie es aus anderen Datenbanksystemen kennen, ist nicht nötig, da die Datei die Datenbank ist. Dies ist bei vielen Embedded-Datenbanken der Fall.

Der SQL-String *createTABLE* ist folgendermaßen aufgebaut. Dem SQL-Befehl *CREATE TABLE* folgt die Angabe des Tabellennamens mit *REMINDER_TABLE*. In der runden Klammer werden nacheinander die einzelnen Spalten definiert. Jede Spaltendefinition muss mindestens aus dem Spaltennamen und dem Datentyp der Spalte bestehen. Weitere Angaben, wie Primary Key oder Not Null, sind optional. Im reinen SQL sieht der Befehl wie folgt aus:

```
CREATE TABLE "reminder" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "filePath" TEXT NOT NULL,
    "subject" TEXT,
    "category" TEXT,
    "type" TEXT,
    "date_time" TEXT,
    "latitude" REAL,
    "longitude" REAL,
    "address" TEXT,
    "sub_address" TEXT,
    "is_new" TEXT,
    "is_near" TEXT,
    CONSTRAINT "FILE_PATH" UNIQUE ("filePath") ON CONFLICT ROLLBACK
);
```

In SQLite kann nur der Primärschlüssel, PRIMARY KEY, die Option *autoincrement* besitzen. Außerdem muss der Datentyp der Spalte ein automatisches Hochzählen ermöglichen. Ein Primärschlüssel einer Tabelle identifiziert jede Tabellenzeile eindeutig innerhalb der Tabelle. Die Option *autoincrement* bewirkt, dass bei jedem Hinzufügen eines Datensatzes in die Tabelle ein interner Zähler um 1 erhöht wird. Das Löschen von Datensätzen aus der Tabelle verändert den Zähler nicht, selbst wenn Sie alle Datensätze mit *delete* löschen, wird beim nächsten Einfügen der Zähler erhöht.

Die Spaltenoption *not null* verlangt, dass der Inhalt dieser Spalte nicht null sein darf. Beim Einfügen oder Aktualisieren von Datensätzen muss diese Spalte immer einen Wert erhalten.

Am Ende des SQL Strings erfolgt die Angabe eines Constraints. Ein Constraint ist eine Zwangsbedingung in der Tabelle, die erfüllt werden muss. In unserer Definition erhält die Spalte *filePath* die Bedingung, dass der Inhalt von *filePath* einzigartig sein muss. Es kann kein identischer zweiter String in dieser Spalte vorkommen, und die Angabe von ROLLBACK bewirkt, dass bei einem Auftreten das Einfügen rückgängig gemacht wird.



PRAXISTIPP: Bei der Erstellung von SQL Statements sollten Sie alle SQL-Schlüsselwörter, wie SELECT, WHERE, DROP usw., komplett in Großbuchstaben notieren, obwohl SQL nicht zwischen Groß- und Kleinschreibung unterscheidet.

Dies ist eine gängige Konvention, und Sie erreichen eine bessere Lesbarkeit.

Listing 10.11 Methode *onUpgrade()* der Klasse DatabaseManager

```
private final static String DROP_TABLE = "DROP TABLE IF EXISTS "
                                         + REMINDER_TABLE;
@Override
public void onUpgrade(SQLiteDatabase db,
                      int oldVersion,
                      int newVersion) {
    db.execSQL(DROP_TABLE);
    onCreate(db);
}
```

Die `onUpgrade()`-Methode aus Listing 10.11 wird ausgeführt, wenn Sie die statische Variable `DB_VERSION` ändern. Als Erstes wird die Tabelle `Reminder` komplett gelöscht, und danach erfolgt der Aufruf der `onCreate()`-Methode für die Tabelle. Der String `DROP_TABLE` enthält den SQL-Befehl zum Löschen der Tabelle. In einer produktiven App sollten Sie dies so nicht durchführen, der User verliert dabei alle in der Tabelle gespeicherten Daten. Sie müssen sich einen Mechanismus überlegen, um die Daten für den Benutzer verfügbar zu machen.

10.2.2 Datensätze hinzufügen, ändern und löschen

Das Hinzufügen, Ändern oder Löschen von Datensätzen ist denkbar einfach. In Listing 10.12 sehen Sie die Methode zum Hinzufügen eines zeitbasierten Reminders. Die Methode nimmt ein `Reminder`-Objekt entgegen und gibt das veränderte `Reminder`-Objekt zurück. Im ersten Schritt der Methode wird eine Referenz auf die beschreibbare Datenbank angelegt. Die nächste Anweisung erstellt das Objekt `values` des Typs `ContentValues`. Die Klasse `ContentValues` ermöglicht die Erstellung eines Datenpakets, das per Key/Value die einzelnen Werte identifiziert. Mit dem ersten Parameter werden die Spaltennamen und mit dem zweiten die Werte übergeben. Dies vereinfacht das Einfügen von Daten erheblich, denn Sie müssen nicht die Reihenfolge der Spalten einhalten. Mit `values.put(...)` fügen Sie die Werte unter Angabe der Spaltenbezeichnung hinzu, z. B. fügt `values.put(SUBJECT, reminder.getSubject())` den Wert des Reminders mit dem Key `Subject` in das Paket ein. Durch `db.insert(...)` wird der Datensatz in die Datenbank eingefügt. Das Insert Statement liefert bei Erfolg die `Id` der neu eingefügten Zeile zurück oder `-1`, wenn der Vorgang fehlgeschlagen ist. Das Insert Statement erwartet die folgenden drei Parameter:

- Tabellennamen
- `nullColumnHack`
- `values`

In SQL ist es nicht möglich, eine völlig leere Zeile ohne Angabe mindestens eines Spaltennamens einzufügen. Mit `nullColumnHack` geben Sie die Spalte an, die ersatzweise im Statement gewählt werden soll, um eine Zeile mit `null` einzufügen. Die Spalte, die hier angegeben wird, muss natürlich `null` Werte erlauben und nicht das Attribut `not null` in der Tabellendefinition haben.

Nachdem der Datensatz mit `db.insert(...)` in die Tabelle eingefügt wurde, wird dem `Reminder`-Attribut `DatabaseId` die von `insert` erhaltene ID zugewiesen, und im Anschluss wird die Verbindung zur Datenbank geschlossen. Zum Abschluss wird das veränderte `Reminder`-Objekt zurückgegeben.



HINWEIS: Sie sollten offene Datenbankverbindungen immer nach der Ausführung schließen, sonst hat dies erhebliche Auswirkungen auf die Performance.

Listing 10.12 Hinzufügen von Daten in die Tabelle

```
private Reminder addNewTimeReminder(Reminder reminder){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(FILE_PATH, reminder.getFilePath());
    values.put(SUBJECT, reminder.getSubject());
    values.put(CATEGORY, reminder.getNoteType().toString());
    values.put(TYPE, reminder.getReminderType().toString());
    values.put(DATE_TIME, getDateTimeString(reminder.getDateTime()));
    long id = db.insert(REMINDER_TABLE, null, values);
    reminder.setDatabaseId(id);
    this.close();
    return reminder;
}
```

Listing 10.13 zeigt die Methode zum Ändern eines Datensatzes in der Datenbank. Um einen Datensatz ändern zu können, muss die Bedingung erfüllt sein, diesen eindeutig in der Tabelle identifizieren zu können. Die Einschränkung in der WhereClause muss sicherstellen, dass nur ein Datensatz betroffen ist, es sei denn, Sie wollen mehrere oder alle ändern. Der Aufbau der Methode ist bis auf das Update Statement `db.update(...)` gleich der Methode `addNewTimeReminder()`. Das Update Statement verlangt vier Parameter in der festgelegten Reihenfolge:

- Tabellennamen
- values
- whereClause
- whereArguments

In der whereClause werden die Spaltennamen angegeben, die zur Einschränkung dienen. Die whereArguments werden als String-Array notiert und müssen in der Reihenfolge der whereClause erscheinen. Sowohl whereClause und whereArguments können null sein, da die Angabe optional ist. Ein Beispiel: `db.update(REMINDER_TABLE, values, null, null)` Wird dieses UpdateStatement so ausgeführt, werden alle Datensätze der Tabelle mit den Werten der Values geändert. Das UpdateStatement erhält als Rückgabewert die Anzahl der Zeilen, die von der Änderung betroffen waren.

Listing 10.13 Datensätze in der Datenbank ändern

```
private Reminder updateTimeReminder(Reminder reminder){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(FILE_PATH, reminder.getFilePath());
    values.put(SUBJECT, reminder.getSubject());
    values.put(CATEGORY, reminder.getNoteType().toString());
    values.put(TYPE, reminder.getReminderType().toString());
    values.put(DATE_TIME, getDateTimeString(reminder.getDateTime()));
    db.update(REMINDER_TABLE, values, TABLE_ID + "=?", 
        new String[]{String.valueOf(reminder.getDatabaseId())});
    this.close();
    return reminder;
}
```

Wie Sie schon bemerkt haben werden, ändern die Methoden aus Listing 10.12 und Listing 10.13 nur Datensätze zeitbasierter Erinnerungen. Die Methoden zum Hinzufügen und

Ändern der ortsbasierten Erinnerungen unterscheiden sich nur bei den *ContentValues*, die verwendet werden.

Das Löschen eines Datensatzes aus der Tabelle ist die einfachste Methode, wie Sie im Listing 10.14 sehen. Zuerst wird wieder eine Referenz auf die beschreibbare Datenbank erstellt und mit *db.delete(...)* der Datensatz gelöscht. Das übergebene Reminder-Objekt enthält das Attribut DataBaseId.

Listing 10.14 Einen Datensatz aus der Tabelle löschen

```
private void deleteReminder(Reminder reminder){
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(REMINDER_TABLE, TABLE_ID + "=?",
              new String[]{String.valueOf(reminder.getDataBaseId())});
    this.close();
}
```

Alle Methoden zum Ändern, Hinzufügen oder Löschen sind *private* Methoden und können nur innerhalb der Klasse *DatabaseManager* verwendet werden. Mit der öffentlichen Methode *saveReminder()* aus Listing 10.15 werden die Reminder in der Tabelle gespeichert, geändert oder gelöscht. Wir haben diesen Ansatz gewählt, um im Code nur eine Methode aufzurufen und uns keine Gedanken darüber machen zu müssen, ob gelöscht, geändert oder neu hinzugefügt werden soll. Der Methode *saveReminder()* wird das Reminder Objekt übergeben, und wir erhalten das geänderte Objekt oder null zurück. Die Methode entscheidet, welche Aktion durchgeführt wird.

Zunächst erfolgt die Erstellung einer Referenz auf die lesbare Datenbank, um mit dem folgenden Cursor die Id und den Reminder-Typ in der Tabelle zu ermitteln.

Die folgende *if*-Anweisung überprüft, ob der Cursor nur einen Datensatz enthält. Ist die Bedingung erfüllt, werden zunächst die Indizes der Spalten im Cursor ermittelt, um im nächsten Schritt die Werte mit *cursor.getString(...)* oder *cursor.getInt(...)*, je nach Datentyp, aus dem Cursor auszulesen. Die ReminderType-Variable *typeInDB* speichert den Wert aus der Datenbank, um diesen später mit dem Typen des Reminder-Objekts zu vergleichen. Da die Datenbank nur String-Objekte speichert und keine Enums, erfolgt die Zuweisung durch *.valueOf(String)*.

Dem übergebenen Reminder-Objekt wird die ermittelte Id der Tabelle zugewiesen, der Cursor geschlossen und die *if*-Anweisung beendet.

Die *else/if*-Anweisung prüft, ob mehr als ein Datensatz in der Tabelle gefunden wurde. Trifft dieser Fall ein, wird das übergebene Reminder-Objekt unverändert zurückgegeben.

Listing 10.15 Methode zum Speichern, Ändern oder Löschen eines Reminders

```
public Reminder saveReminder(Reminder reminder){
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.query(REMINDER_TABLE,
                            new String[]{String.valueOf(TABLE_ID), TYPE},
                            FILE_PATH+"=?",
                            new String[]{reminder.getFilePath().toString()},
                            null, null, null);
    if(cursor.getCount() == 1){
        ReminderType typeInDB = null;
        cursor.moveToFirst();
        int idIndex = cursor.getColumnIndex(TABLE_ID);
```

```

int typeIndex = cursor.getColumnIndex(TYPE);
int id = cursor.getInt(idIndex);
String type = cursor.getString(typeIndex);
typeInDB = ReminderType.valueOf(type);
reminder.set DataBaseId(id);
cursor.close();
} else if(cursor.getCount() > 1){
    return reminder;
}
switch (reminder.getReminderType()){
    case time:
        if(reminder.get DataBaseId() < 0 ){
            reminder = addNewTimeReminder(reminder);
        }else if (reminder.get DataBaseId() > 0
                  && typeInDB == reminder.getReminderType()){
            reminder = updateTimeReminder(reminder);
        }else if (reminder.get DataBaseId() > 0
                  && typeInDB != reminder.getReminderType()){
            deleteReminder(reminder);
            reminder = addNewTimeReminder(reminder);
        }
        break;
    case location:
        break;
    case none:
        if(reminder.get DataBaseId() > 0 ){
            deleteReminder(reminder);
            reminder = null;
        }
        break;
    default: break;
}
return reminder;
}

```

10.2.3 Alle Datensätze einer Tabelle auslesen

Die Methode `getAllReminder()` aus Listing 10.16 liefert alle Datensätze der Tabelle reminder als typisierte ArrayList<Reminder> zurück.

Die Funktionsweise der `getAllReminder(...)`-Methode sollte Ihnen kein Kopfzerbrechen bereiten. Bis auf die *do-while*-Schleife, sind Ihnen alle Funktionsprinzipien bekannt.

Es wird eine Referenz auf die lesbare Datenbank erstellt, die ArrayList `reminderList` wird initialisiert, und es wird ein Cursor erstellt, der direkt mit der Datenbank-Query befüllt wird. Enthält der Cursor keine Daten, wird die Methode beendet und `null` zurückgegeben.

Nachdem der Cursor mit `moveToFirst()` an die erste Position gesetzt wurde, werden die Spalten-Indizes der Ergebnisliste ermittelt und in Variablen zwischengespeichert sowie ein Reminder-Objekt mit `null` initialisiert.

Die *do-while*-Schleife ist eine so genannte fußgesteuerte Schleife. Das bedeutet, der Code in der Schleife wird immer mindestens einmal ausgeführt und danach die `while`-Anweisung. Dies ist notwendig, da der Cursor sich schon an der ersten Position in der Ergebnisliste befindet.

In der *do-while*-Schleife werden zuerst die Werte aus dem Cursor ausgelesen und in entsprechenden Variablen zwischengespeichert. Es folgt die Erstellung eines Reminder-Objekts,

das als Parameter die Werte in den vorher gespeicherten Variablen übergibt. Im Weiteren werden dem Reminder-Objekt noch die Attribute DatabaseID und DateTime übergeben, wobei die Variable dateTime dahingehend überprüft wird, ob diese leer ist. Bei der Zuweisung von DateTime des Reminders erfolgt die Umwandlung des Strings durch die `getDateTimeLong()`-Methode in den Datentyp *Long*, die das Datum in Millisekunden zurückgibt.

Die `while`-Anweisung der Schleife positioniert den Cursor nun auf der nächsten Zeile der Ergebnisliste, so lange, bis die Liste vollständig durchlaufen ist.

Nach Verlassen der *do-while*-Schleife wird zunächst der Cursor geschlossen und am Ende die gefüllte Liste mit den Reminder-Objekten an den Aufrufer zurückgegeben.

Listing 10.16 Alle Datensätze einer Tabelle als Liste ausgeben

```
public List<Reminder> getAllReminder(){
    SQLiteDatabase db = this.getReadableDatabase();
    List<Reminder> reminderList = new ArrayList<Reminder>();
    Cursor cursor = db.query(REMINDER_TABLE, null,
                           null, null, null, null, null);
    if(cursor.getCount() < 1){
        return null;
    }
    cursor.moveToFirst();
    int idIndex = cursor.getColumnIndex(TABLE_ID);
    int filePathIndex = cursor.getColumnIndex(FILE_PATH);
    int subjectIndex = cursor.getColumnIndex(SUBJECT);
    int categoryIndex = cursor.getColumnIndex(CATEGORY);
    int dateTimeIndex = cursor.getColumnIndex(DATE_TIME);
    Reminder reminder =null;
    do{
        long id = cursor.getInt(idIndex);
        String filePath = cursor.getString(filePathIndex);
        String subject = cursor.getString(subjectIndex);
        String category = cursor.getString(categoryIndex);
        String dateTime = cursor.getString(dateTimeIndex);
        reminder = new Reminder(filePath, subject,
                               NoteType.valueOf(category));
        reminder.setDatabaseId(id);
        if(!dateTime.isEmpty()){
            reminder.setDateTime(getDateTimeLong(dateTime));
        }
        reminderList.add(reminder);
    }while(cursor.moveToNext());
    cursor.close();
    return reminderList;
}
```

10.2.4 Datenbankzugriff per Kommandozeile oder Eclipse Plug-in

Bei der Entwicklung von Datenbankanwendungen möchte man oft die Ergebnisse außerhalb des Java-Quellcodes kontrollieren bzw. Einträge ändern oder hinzufügen, und das Ergebnis in der App analysieren. Das ADT-Plug-in bietet hier die Möglichkeit, mit dem Kommandozeilen-Programm *sqlite3* auf die Datenbank zuzugreifen. Auf der Konsole können Sie nun alle von SQLite unterstützten SQL-Kommandos nutzen (Bild 10.4).

```
D:\>adb shell
root@android:/ # sqlite3 /data/data/eu.scytec.notes/databases/notesReminder.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"

sqlite> select * from reminder;
2|/storage/sdcard0/ScytecNotes/txt/work.txt|work|text|time|2013-06-04 22:12|||||
3|/storage/sdcard0/ScytecNotes/txt/shopping.txt|shopping|text|time|2013-06-04 22:13|||||
4|/storage/sdcard0/ScytecNotes/txt/private.txt|private|text|time|2013-06-04 22:19|||||
sqlite>.exit
.root@android:/ # exit
exit

D:\>
```

Bild 10.4 Zugriff auf die Datenbank mit sqlite3 in der Konsole

Eine weitere Möglichkeit bietet das Eclipse-Plug-in *sqlitexmlbrowser*² bzw. *questoid*³. Dieses Plug-in ist unter beiden Bezeichnungen im Netz zu finden. Die Benutzung ist denkbar einfach. Als Erstes müssen Sie das Plug-in (*.jar*-Datei) in den Unterordner *Plug-ins* von Eclipse einfügen. Nach einem Neustart von Eclipse wechseln Sie in die DDMS-Ansicht und markieren im File Explorer die SQLite-Datenbankdatei. Anschließend öffnen Sie die Datenbank durch Klick auf das Datenbanksymbol, wie in Bild 10.5 zu sehen.

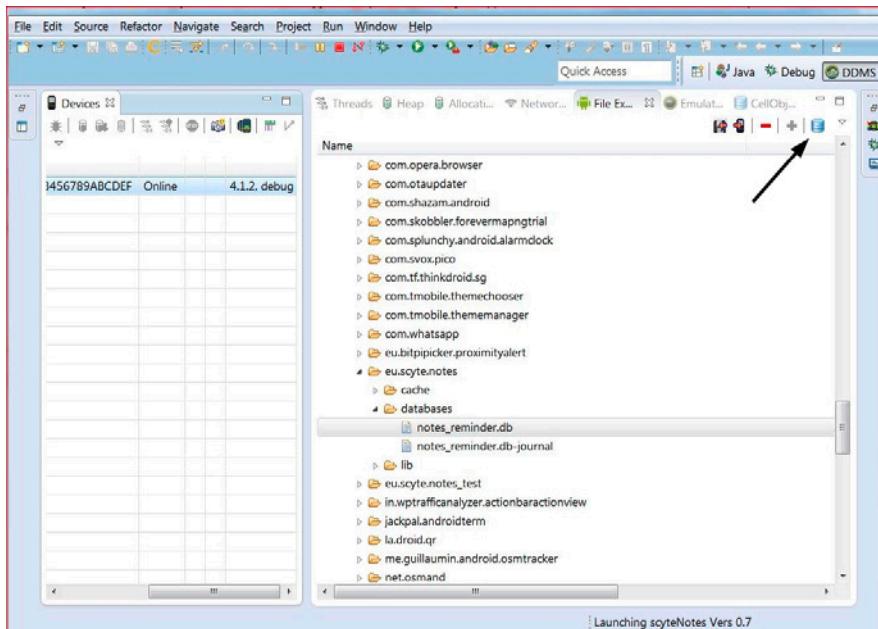


Bild 10.5 File Explorer von Eclipse mit Plug-in in der DDMS-Perspektive

² <http://code.google.com/p/cellobjekt/>

³ <http://www.java2s.com/Code/Jar/c/Downloadcomquestoidsqlitebrowser120jar.htm>

■ 10.3 Daten aus der Datenbank anzeigen

Um die Reminder-Daten auch in der Liste anzeigen zu können, sind nur ein paar Änderungen nötig. Wir übergeben die Anzeigedaten der Reminder den Notiz-Objekten. Dazu müssen Sie die Klasse Note um das Attribut *mReminderText* des Typs String und die Getter- und Setter-Methoden für das Attribut in der Klasse erstellen.

Um den Reminder-Text den Note-Objekten übergeben zu können, muss die Methode *getReminderFromList()* aus Listing 10.17 der Klasse *MainActivity* hinzugefügt werden. Der Methode wird der Dateipfad des Note-Objekts übergeben, dann durchläuft sie die Liste mit den Reminder-Objekten und überprüft in der *if*-Anweisung den Pfad des Reminder-Objekts mit dem übergebenen String. Ein String-Vergleich sollte mit der *String.equals()*-Methode durchgeführt werden und nicht mit *==*. Diese liefert *true*, wenn der Inhalt beider Strings gleich ist.

Wurde ein passendes Reminder-Objekt gefunden, liefert diese das Reminder-Objekt zurück, sonst **null**.

Listing 10.17 Reminder der Notiz ermitteln und Anzeigetext zurückgeben

```
private Reminder getReminderFromList(String filePath){  
    if(!listReminder.isEmpty()) {  
        for(Reminder reminderItem: listReminder) {  
            if(reminderItem.getFilePath().equals(filePath)) {  
                return reminderItem;  
            }  
        }  
    }  
    return null;  
}
```

Der Aufruf der zuvor beschriebenen Methode erfolgt in der Methode *getFiles()* der *Main Activity*-Klasse, die jetzt noch angepasst werden muss (Listing 10.18).

Zunächst wird eine Referenz auf die Klasse *DatabaseManager* erstellt. Die als Klassenvariable deklarierte Liste *listReminder* wird durch die Methode *.getAllReminder()*, die eine *ArrayList* mit den Reminder-Objekten liefert, gefüllt.

In der *for*-Schleife erhält *reminderItem* den Rückgabewert der *getReminderFromList()*-Methode, dem als Parameter der absolute Dateipfad des File-Objekts übergeben wird, mit dem zuvor das Note-Objekt erstellt wurde. Dieser ist einmalig im Dateisystem und auch in der Datenbank. Wurde ein Reminder-Objekt von der Methode zurückgegeben, ist *reminder Item* nicht **null**, und dem Attribut *ReminderText* des *noteItems* wird mit der *.setReminderText(...)*-Methode der Rückgabewert der *getReminderText(...)*-Methode aus Listing 10.19 übergeben. Diese Methode beschreiben wir nicht weiter.

Damit der nun im Note-Objekt vorhandene *ReminderText* im *ListItem* der *ListView* angezeigt wird, müssen Sie in der Methode *setNoteItem(...)* der Klasse *ListItemView* dem *TextView*-Element des Reminders mit *.setText(note.getReminderText())* den Text übergeben.

Listing 10.18 Änderungen der Methode getFiles() der MainActivity

```
DatabaseManager dbManager = new DatabaseManager(this);
listReminder = dbManager.getAllReminder();
Reminder reminderItem;
for (File file : files) {
    noteItem = new Note(file);
    reminderItem = getReminderFromList(file.getAbsolutePath());
    if(reminderItem != null){
        noteItem.setReminderText(
            "+getReminderText(reminderItem)+\"");
    } else{
        noteItem.setReminderText(getString(R.string.no_reminder));
    }
    listNotes.add(noteItem);
```

Listing 10.19 Methode getReminderText() der Klasse MainActivity

```
private String getReminderText(Reminder reminder){
    String text = "no reminder";
    if(reminder != null ){
        switch (reminder.getReminderType()){
            case time:
                if(reminder.getDateTime() > 0){
                    Date dt = new Date(reminder.getDateTime());
                    text = Date_Format.format(dt).toString()
                    + " " + Time_Format.format(dt);
                }else {
                    text = "Error cast DateTime";
                }
                break;
            case location:
                text = "location";
                break;
            default: break;
        }
    }else {
        text = "no reminder";
    }
    return text;
}
```

■ 10.4 Zwischenstand der App (Version 0.7)

In diesem Kapitel zeigten wir am Beispiel der zeitbasierten Erinnerung, wie Sie Daten in eine Datenbank einfügen, ändern oder löschen. Diese Erinnerungen sind jetzt in der Datenbank vorhanden, zeigen jedoch noch keine Wirkung.

Wie ortsbasierte Erinnerungen erstellt, in die Datenbank eingefügt und die Position in der Karte angezeigt werden, zeigen wir in Kapitel 11.

11

Google Maps Api V2 und LocationService

In diesem Kapitel wollen wir Ihnen zeigen, wie Sie Ihren aktuellen Standort in einer Google Maps-Karte darstellen und der Karte einen eigenen Marker hinzufügen können, um später eine ortsbasierte Erinnerung zu erstellen. Das Layout soll einem Dialog entsprechen.

Wir haben uns entschieden, die Maps-Karte in einer eigenen Activity – und nicht wie bei zeitbasierten Erinnerungen, in einem eigenen Dialog – darzustellen. Die Gründe für die Entscheidung sind relativ einfach. Die Nutzung der Karte ist in einem Dialog sehr unübersichtlich, und selbst wenn das Layout des Dialogs vergrößert werden würde, ist die Darstellung immer noch zu klein, als dass der Benutzer diese als angenehm empfinden würde.

■ 11.1 Vorbereitungen zur Verwendung von Google Maps Api V2

Mit der Einführung der zweiten Version der Google Maps Api ist für Entwickler vieles einfacher geworden. Die Erstellung von Pins und Overlays war recht umständlich, die Darstellung der Map für unterschiedliche Geräte nur über Umwege möglich. Diese Probleme wurden mit der Maps Api-Version 2 abgestellt. Mit der Version 2 wird die Google Maps Api als Teil der Google Play Services bereitgestellt. Um die Google Play Services nutzen zu können, muss die Bibliothek der Google Play Services in der Entwicklungsumgebung und auf dem Gerät der Google Play Store vorhanden sein. Die Bibliothek können Sie Ihrer Umgebung einfach über den SDK Manager des ADT-Plug-ins hinzufügen (Bild 11.1).

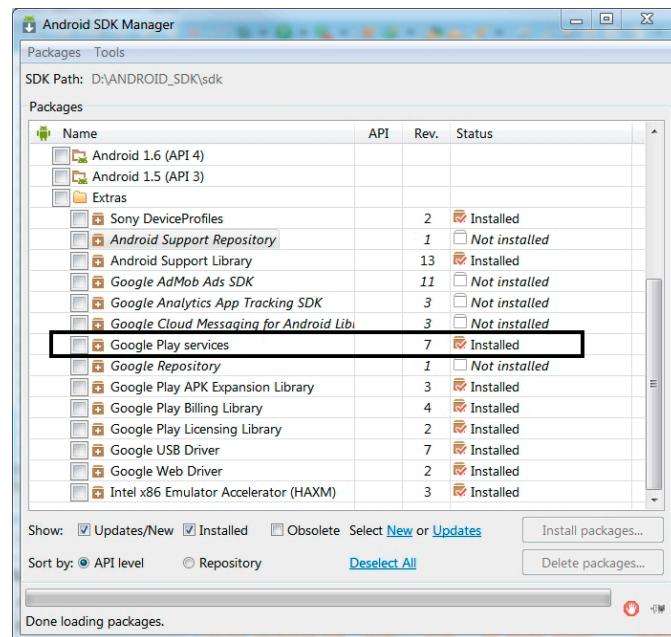


Bild 11.1 SDK Manager-Installation der Google Play Services-Bibliothek

Nachdem Sie die Bibliothek erfolgreich installiert haben, müssen Sie ein neues Android-Projekt erstellen, jedoch aus bestehendem Code. Dieses Mal wählen Sie in Eclipse NEW PROJECT und im folgenden Assistenten ANDROID > ANDROID PROJECT FROM EXISTING CODE, wie im Bild 11.2 zu sehen.

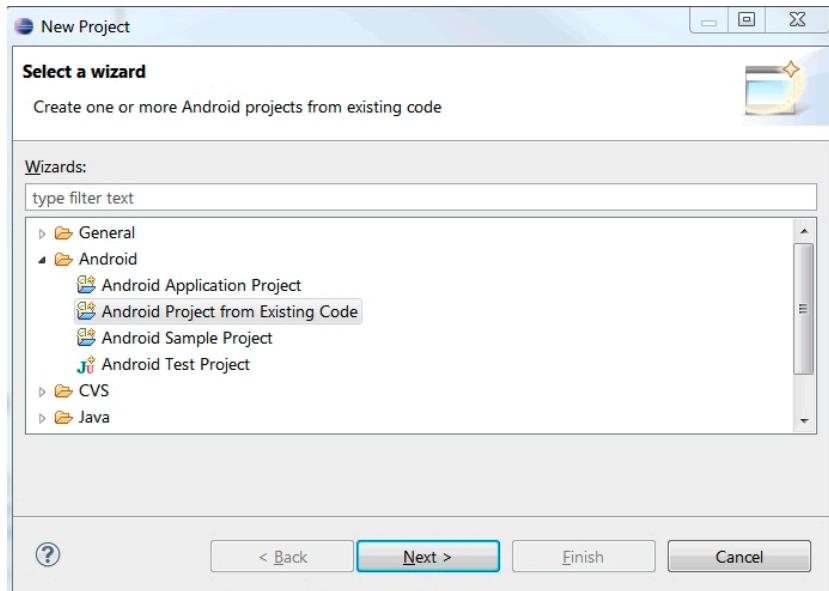


Bild 11.2 Projekt aus bestehendem Code erstellen

Das Projekt der Google Play Services-Bibliothek finden Sie im Android SDK-Ordner ... \sdk\extras\google\google_play_services\libproject\google-play-services_lib (Bild 11.3). Beim automatischen Erstellen des Projekts der Google Play Services-Bibliothek erhalten Sie eine Fehlermeldung, wenn Sie das SDK des Api Levels 8 nicht installiert haben, und die Bibliothek wird nicht erstellt. Sie können das Problem einfach beheben, in dem Sie entweder das SDK installieren oder im Manifest der Bibliothek die Einstellung des Min-SDK an Ihre Umgebung anpassen.

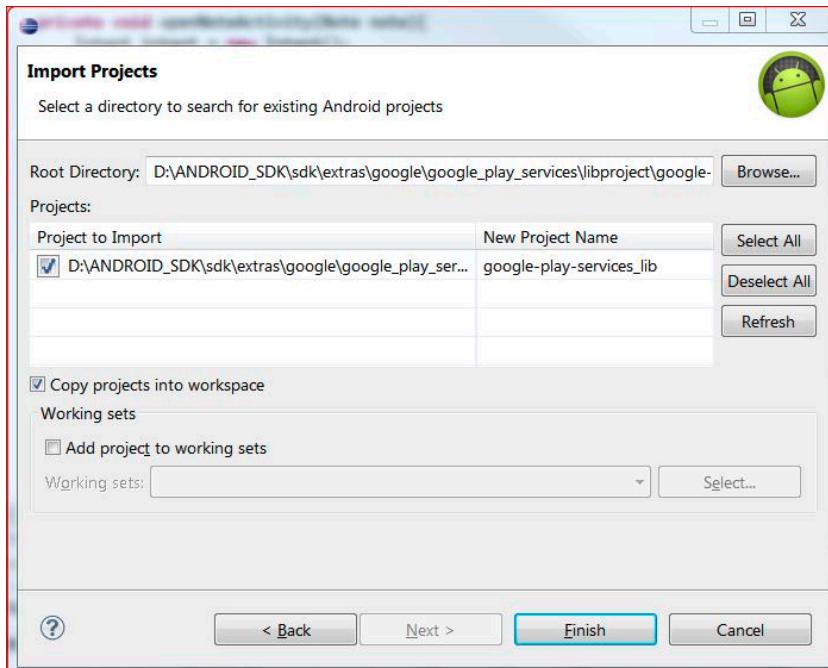


Bild 11.3 Google Play Services-Bibliothek als eigenes Projekt hinzufügen

Zum Abschluss müssen Sie noch die Bibliothek dem Projekt hinzufügen. Markieren Sie dazu den Projektordner *scytenotes* und wählen im Kontextmenü **PROPERTIES** *Android*. Erstellen Sie nun durch ADD eine Referenz auf die Google Play Services-Bibliothek im Projekt. Da im Workspace nur eine Bibliothek vorhanden ist, erhalten Sie nur diese im folgenden Dialog zur Auswahl. Aktivieren Sie das Projekt der Bibliothek und die Option *Copy projects into workspace* und schließen den Vorgang mit FINISH ab. Das Projekt der Bibliothek wird in den Workspace von Eclipse kopiert. Sie können nun Änderungen vornehmen, ohne das Original zu verändern.

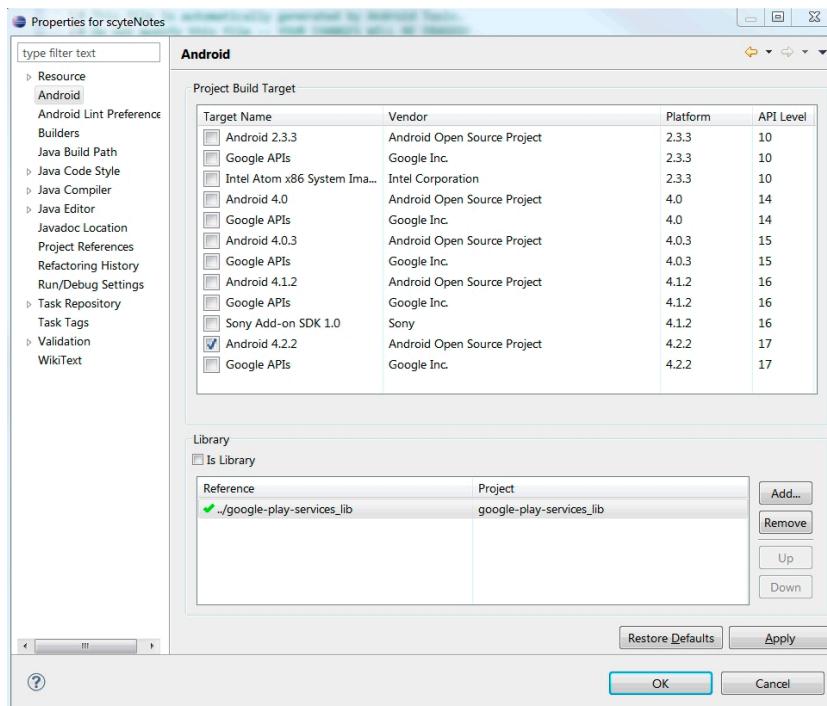


Bild 11.4 Bibliothek im Projekt *scyte.notes* hinzugefügt

Nach Hinzufügen der Bibliothek kommt es immer wieder einmal vor, dass im Projekt Fehler angezeigt werden, und dieses deshalb nicht mehr kompiliert werden kann. Häufig werden alle View-Elemente rot unterstrichen, und Sie werden aufgefordert, *android.R* zu importieren. Ursache für dieses Verhalten ist, dass der Builder die Bibliothek nicht richtig einbinden kann. Oft reicht es aus, den Pfad der Bibliothek in den Projekteinstellungen (*projekt.properties*) anzupassen, im Standard ist dies die *android.library.reference.1=../google-play-services_lib*. Geben Sie hier, wenn nötig, den absoluten Pfad zur Bibliothek an, öffnen Sie im Explorer den Workspace-Ordner und ermitteln den Pfad zum Unterordner */libs* des Google Play Services-Projekts. Eine weitere Möglichkeit ist, dass Sie die Einstellungen von *Java Build Path* anpassen, die Reihenfolge ändern und *Android Dependencies* aktivieren, wie in Bild 11.5 zu sehen. Diese Maßnahmen haben bisher fast immer die Probleme gelöst.

Sie haben nun die Bibliothek erfolgreich in das Projekt eingebunden, eventuelle Probleme behoben, jedoch sind noch einige Maßnahmen notwendig, damit Sie die Google Maps Api in Ihren Projekten verwenden können.

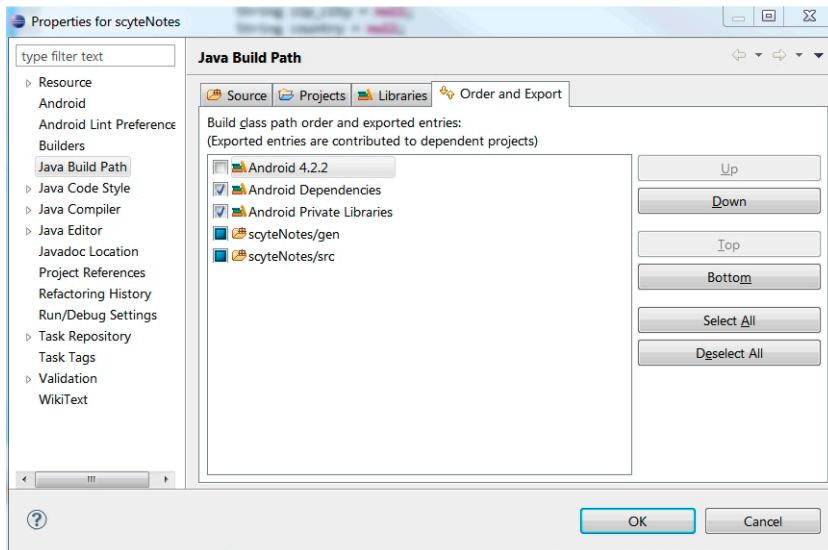


Bild 11.5 Build Path-Einstellungen des Projekts ändern

11.1.1 API-Key und Rechte im Manifest anpassen

Wir setzen voraus, dass Sie einen gültigen Entwicklerzugang bei Google haben. Falls nicht, finden Sie unter <https://support.google.com/googleplay/android-developer/> eine Anleitung zur Registrierung. Die Registrierung ist nicht kostenlos, Google erhebt eine Gebühr in Höhe von derzeit 25 US-Dollar.



HINWEIS: Sie müssen sich nicht zwingend als Entwickler registrieren, um die Google Maps Api in Ihrer App verwenden zu können, solange der Paketname `eu.scyte.notes` ist.

Um die Google Maps Api in einem eigenen Projekt verwenden zu können, benötigen Sie als Entwickler einen Api-Key, der das Package der App mit Ihrem Entwicklerkonto verknüpft. Dieser Api-Key muss im Manifest der App eingetragen werden. Außerdem benötigt die App verschiedene Rechte. Sie muss auf das Internet zugreifen können, um die Daten zu laden, und auf den LocationService zugreifen, um die Position zu bestimmen.

Den Api-Key erstellen Sie in der Google Api Console (<https://code.google.com/apis/console/>). Voraussetzung für die Nutzung der Google Api Console ist natürlich, dass Sie als Entwickler registriert sind.

Zur Erstellung eines Google Maps Api Keys benötigen Sie einen SHA1-Fingerabdruck, den Sie mit `keytool` aus dem JDK erstellen. Mit dem JDK `keytool` können Sie JAVA .jar-Dateien signieren sowie Zertifikate in einem `Keystore` verwalten oder selbstsignierte erstellen. Das Key-Tool befindet sich im `bin`-Ordner des JDK. Haben Sie den Pfad zum JDK in den Windows-Systemvariablen eingetragen, können Sie es direkt in der Konsole ausführen.

Mit dem ADT-Plug-in in Eclipse können Sie Ihre Apps beim Exportieren zusätzlich signieren. In der Google Api Console erstellen Sie eine Verknüpfung zu einem Zertifikat in Ihrem Keystore über dessen SHA1-Fingerabdruck und den Package-Namen Ihrer App. Der Debug-Keystore wird automatisch erzeugt, wenn Sie das erste Mal eine App erstellen und auf einem Gerät oder im Emulator ausführen. Dabei wird ein Debug-Zertifikat im Debug-Keystore abgelegt. Der Debug-Keystore, *debug.keystore*, befindet sich im Ordner *.android* unterhalb des Windows-Benutzer-Verzeichnisses *C:\Benutzer\IHR BENUTZER-NAME*.

Bei der Erstellung des Maps Api Keys sollten Sie für jede App ein eigenes Projekt in der Api-Console anlegen und sich mit demselben Google-Konto anmelden, mit dem Sie auch die App im Play Store veröffentlichen. In der Vergangenheit gab es immer wieder Probleme mit Google-Konten, die ihre eigene Domain verwenden oder @*gmail.com* lauten.

Google Maps Api-Key erstellen

Am Beispiel des Debug-Zertifikats erläutern wir die Erstellung des Google Maps Api-Keys.

1. Melden Sie sich in der Google Api Console an (<https://code.google.com/apis/console>) und aktivieren Sie unter Services *Google Maps Android API v2*.
2. Geben Sie in der Windows-Konsole folgenden Befehl ein: `keytool -v -list -alias androiddebugkey -keystore C:\Users\Android\.android\debug.keystore`, **Passwort: android**.

```

Eingabeaufforderung
Microsoft Windows [Version 6.1.7600]
Copyright <C> 2009 Microsoft Corporation. Alle Rechte vorbehalten.
C:\Users\Android>keytool -v -list -alias androiddebugkey -keystore C:\Users\Android\.android\debug.keystore
Keystore-Kennwort eingeben:
Aliasname: androiddebugkey
Erstellungsdatum: 22.01.2013
Eintragstyp: PrivateKeyEntry
Zertifikatkettenglänge: 1
Zertifikatversion: 3
Eigent*ner: CN=Android Debug, O=Android, C=US
Aussteller: CN=Android Debug, O=Android, C=US
Seriennummer: 50ff18f2
Gültig von: Tue Jan 22 23:55:46 CET 2013 bis: Thu Jan 15 23:55:46 CET 2043
Zertifikat-Fingerprints:
    MD5: 2D:DC:34:0E:08:34:C7:9D:7F:85:82:43:C9:F0:B0:D7
    SHA1: DF:5B:47:0B:72:A4:47:3B:27:9D:9E:9A:5F:BD:CC:61:E5:DB:A5:01
    SHA256: FB:B5:8B:2E:09:3A:E1:42:07:68:9E:D4:D8:09:32:52:7F:97:CE:0C:80:CD:B4:B5:80:8A:37:A7:8A:25:51:23
    Signaturalgorithmusname: SHA1withRSA
    Version: 3

C:\Users\Android>_

```

Bild 11.6 Ausgabe keytool in der Kommandozeile

3. Öffnen Sie die Google Api Console und erstellen mit CREATE NEW ANDROID KEY... einen neuen Key. Geben Sie in das Fenster den **SHA1-Fingerprint** und den Namen des **Packages** ein.

The screenshot shows the 'Simple API Access' section of the Google API Console. It displays a generated API key, its SHA1 fingerprint, activation date, and email address. Below the key details are four buttons: 'Create new Server key...', 'Create new Browser key...', 'Create new Android key...', and 'Create new iOS key...'. At the bottom, there's a section titled 'Notification Endpoints'.

Key for Android apps (with certificates)

API key: AIzaSyBwthuHJoLl5t0Er6-w2hQtZxxI8TsVzvM
 Android apps: DE:5B:47:8A:72:A4:47:3B:27:9D:9E:9A:5F:BD:CC:61:E5:DB:A5:01;eu.scyte.notes
 Activated on: Apr 24, 2013 7:31 AM
 Activated by: [REDACTED]@gmail.com

Create new Server key... Create new Browser key... Create new Android key... Create new iOS key...

Notification Endpoints

Bild 11.7 Google Maps Api-Key vom Package eu.scyte.notes

Sie können jedem Api-Key mehrere Zertifikate zuordnen, derzeit ist nur das Debug-Zertifikat über den SHA1-Fingerprint zugeordnet. Bei der Veröffentlichung der App im Play Store signieren wir die Apk-Datei und werden den SHA1-Fingerprint hinzufügen.

Android-Manifest anpassen:

Um Google Maps-Karten in der App anzeigen zu können, muss die App auf das Internet zugreifen, die LocationProvider Network und GPS verwenden und den Netzwerkstatus abfragen können. Die Sicherheitseinschränkungen von Android verlangen, diese Rechte einer Anwendung explizit zu gewähren, und der User wird vor dem Download der App über dieses Recht informiert.

Die Genehmigung in Listing 11.1 kann die App Maps-Daten empfangen und legt den Sicherheitslevel fest.

Listing 11.1 Eigene Map-Berechtigung des Pakets eu.scyte.notes

```
<permission
    android:name="eu.scyte.notes.maps.permission.MAPS_RECEIVE"
    android:protectionLevel="signature" />
```

Die Darstellung der Karten verlangt Unterstützung durch die OpenGL ES-Grafikbibliothek in der Version 2. Die Definition der Anforderung ist in Listing 11.2 dargestellt. Der Eintrag *uses-feature* bewirkt auch, dass alle Geräte, die diese Funktionen nicht unterstützen, im Google Play Store gefiltert werden, und diese App nicht zur Installation angeboten bekommen.

Listing 11.2 Uses-Features im Android-Manifest

```
<uses-feature android:glesVersion="0x00020000"
    android:required="true"/>
```

Listing 11.3 Uses-Permissions im Android-Manifest

```
<uses-permission android:name=
    "com.google.android.providers.gsf.permission.READ_GSERVICES" />
```

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

Der Eintrag des Api-Keys aus Listing 11.4 im Manifest erfolgt im Tag `<application>` nach den Einträgen der Activities.

Listing 11.4 Api-Key im Android-Manifest

```
01 <meta-data android:name="com.google.android.maps.v2.API_KEY"
           android:value="AIzaSyBwthuHJoL15t0Er6-w2hQtZxxI8TsVzvM">
</meta-data>
```



HINWEIS: Wenn Sie das Paket der App nicht anders benannt haben, verwenden Sie den API-Key aus Listing 11.4.

■ 11.2 Ortsbestimmung mit dem LocationService

Es stellt sich jetzt natürlich die Frage, wie die aktuelle Position bestimmt wird. Sie haben unter Android verschiedene Möglichkeiten Ihre Position zu bestimmen.

Die Klasse *LocationManager* bietet z.B. die Möglichkeit, mit den LocationService durch Verwendung des eingebauten GPS-Sensors (soweit im Gerät vorhanden) und über den Netzwerk-Provider, die Position zu bestimmen.

Da wir die aktuelle Position in Google Maps-Karten anzeigen wollen, müssen folgende Bedingungen erfüllt sein:

- Es muss eine Verbindung zum Internet bestehen.
- Der Client muss sich mit den Play Services verbinden können.
- Mindestens ein LocationProvider, GPS oder Network muss verfügbar sein.

Sind diese Bedingungen nicht erfüllt, ist die Verwendung der Google Maps-Karten nicht möglich.

Um die Überprüfungen durchzuführen, bevor die *LocationMapsActivity* aufgerufen wird, haben wir jeweils eine Methode zur Überprüfung der Bedingung in die eigene Klasse *MapsHelper* implementiert.

Wird die Klasse *LocationHelper* (Listing 11.5) in einer anderen Klasse initialisiert, muss der Kontext dem Konstruktor als Parameter übergeben werden. Bei der Initialisierung wird sofort eine Referenz auf den LocationService durch den LocationManager erstellt. Danach erfolgt der Aufruf der Methode *getLocationProvider()* (Listing 11.6), die als Parameter den

Wert von `mLocationManager` übergibt und als Rückgabewert den LocationProvider als String erhält. Ist der zurückgelieferte Wert der Methode nicht `null` erfolgt mit der Methode `.getLastKnownLocation()` die Ermittlung der letzten bekannten Position.

Listing 11.5 Signatur und Konstruktor der Klasse LocationHelper aus dem Paket eu.scyte.notes.data

```
public class LocationHelper {
    String mLocationProvider;
    Location mLocation;
    LocationManager mLocationManager;
    Context mContext;

    public LocationHelper(Context context){
        mContext = context;
        mLocationManager = (LocationManager)
            mContext.getApplicationContext()
            .getSystemService(Context.LOCATION_SERVICE);
        mLocationProvider = getLocationProvider(mLocationManager);
        if(mLocationProvider != null){
            mLocationManager.requestLocationUpdates(mLocationProvider,
                500, 1, mLocationListener);
            mLocation =mLocationManager
                .getLastKnownLocation(mLocationProvider);
        }
    }
}
```

Mit der Methode `getLocationProvider()` wird der Provider zur Standortbestimmung anhand von Kriterien ermittelt. Die Klasse `Criteria` aus dem Paket `android.location` stellt öffentliche Methoden bereit, um Kriterien festzulegen, und durch die Methode `getBestProvider()` diese anhand der zuvor festgelegten Methoden auszuwählen. Dieses Vorgehen erspart die einzelne Prüfung, welche Provider zur Verfügung stehen, und man kann programmäßig Kriterien festlegen, die erfüllt werden müssen. In dieser Methode werden folgende Kriterien zur Ermittlung des Providers festgelegt: mit `setAccuracy()` die Genauigkeit, mit `.setAltitudeRequired()`, ob die Höhe erforderlich ist, mit `.setCostAllowed()`, ob der Provider Kosten verursachen darf (das ist beim Netzwerkprovider der Fall, wenn die Verbindung über GPRS, UMTS bzw. LTE aufgebaut wird), und mit `setPowerRequirement()` der Stromverbrauchs des Provider.

Listing 11.6 Methode, die den Namen des LocationProviders als String zurück liefert

```
private String getLocationProvider(LocationManager lm) {
    String provider = null;
    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_MEDIUM);
    criteria.setAltitudeRequired(false);
    criteria.setCostAllowed(true);
    criteria.setPowerRequirement(Criteria.POWER_MEDIUM);
    provider = lm.getBestProvider(criteria, true);
    return provider;
}
```

Die Methode aus Listing 11.7 liefert `true` zurück, wenn eine Netzwerkverbindung besteht. Die Klasse `ConnectivityManager` überwacht den Status von Wi-Fi, GPRS, UMTS usw. und sendet Broadcast-Nachrichten, wenn der Status sich verändert.

Listing 11.7 Methode zum Überprüfen der Internetverbindung

```
public boolean checkNetworkIsOnline() {
    ConnectivityManager cm = (ConnectivityManager)
        mContext.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo netInfo = cm.getActiveNetworkInfo();
    if (netInfo != null && netInfo.isConnectedOrConnecting()) {
        return true;
    }
    return false;
}
```

Die Methode *checkLocationService()* liefert die Anzahl der verfügbaren Provider zur Standortbestimmung. Ist kein Provider aktiviert, ist der Rückgabewert = 0, ist nur ein Provider vorhanden, ist der Wert = 1 und sind sowohl NETWORK und GPS vorhanden, ist der Rückgabewert = 2. Der Aufruf der Methode erfolgt in der Klasse Note.

Listing 11.8 Methode zum Überprüfen der verfügbaren LocationProvider

```
public int checkLocationService(){
    int result =0;
    if(mLocationManager.isProviderEnabled
        (LocationManager.NETWORK_PROVIDER)){
        result += 1;
    }
    if(mLocationManager.isProviderEnabled
        (LocationManager.GPS_PROVIDER)){
        result +=1;
    }
    return result;
}
```

Listing 11.9 Methode zur Rückgabe der Position als Paar von Latitude und Longitude

```
public LatLng getLatLng() {
    LatLng latLng = null;
    if (mLocation != null) {
        latLng = new LatLng(mLocation.getLatitude(),
            mLocation.getLongitude());
    } else {
        mLocationManager.requestLocationUpdates
            (mLocationProvider, 500, 1, mLocationListener);
        latLng = new LatLng(0, 0);
    }
    return latLng;
}
```

■ 11.3 Eigene Activity for Result für die Map

Die Darstellung von Kartendaten in einem Dialog ist nicht sehr praktikabel. Durch Klick auf die Karte geben wir dem Nutzer die Möglichkeit, einen eigenen Marker hinzuzufügen, um die ortsbasierte Erinnerung eines selbst gewählten Standorts einzurichten.

Wird die *LocationMapsActivity* aufgerufen, wird eine Karte mit dem aktuellen Standort des Users als roter Marker mit Adresse angezeigt. Je verfügbaren und genutzten Provider kann der Standort der Karte vom tatsächlichen Standort erheblich abweichen. Der Benutzer hat die Möglichkeiten, die Karte zu verschieben und zu zoomen. Mit einem Klick in die Karte wird ein neuer blauer Marker hinzugefügt und nun in diesem Marker die Adresse angezeigt. Dieser blaue Marker kann durch einen langen Klick zum Verschieben aktiviert und nun an eine beliebige Position innerhalb der Karte verschoben werden. Während des Verschiebens wird die Adresse der aktuellen Position des Markers angezeigt. Der Benutzer hat somit bei einem sehr ungenauen Standort die Möglichkeit, diesen besser festzulegen oder eine ortsbasierte Erinnerung für einen bestimmten, fernen Ort einzurichten. Wie auch bei der Verwendung von Google Maps ist die Usability von der Netzverbindung abhängig, da die Kartendaten von den Maps-Servern geliefert werden. Hat der User einen eigenen Marker hinzugefügt, werden die Positionsdaten und Adressdaten des User-Markers (blau) in die Datenbank übernommen. Ist kein User-Marker vorhanden, erfolgt die Übernahme der Daten vom Start-Marker (rot). Bei Betätigung des OK-Buttons werden die Daten übernommen und an die Activity, die die *LocationMapsActivity* aufgerufen hat, zurückgegeben.

MapFragment im Layout

Das Layout der *LocationMapsActivity* ist relativ einfach aufgebaut. Es besteht aus dem MapFragment und einer Tabelle in der Fußleiste, welche die zwei Buttons OK und CANCEL enthält (siehe Bild 11.8).

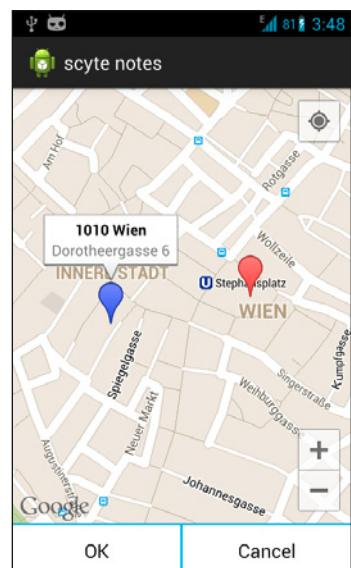


Bild 11.8
Ansicht der LocationMapsActivity

Listing 11.10 enthält den XML-Code, mit dem Sie das Map-Fragment dem Layout hinzufügen. Das Attribut *android:name* verweist auf die Klasse des Fragments, alternativ können Sie auch das Attribut *class* verwenden. Wir verwenden das *MapFragment*, das allerdings nur ab ICS eingesetzt werden kann. Um frühere Versionen von Android zu unterstützen, können Sie das *SupportFragment* nutzen. Jedoch müssen Sie zusätzlich noch die *Support Library* verwenden, da frühere Versionen vor ICS im Standard keine Fragments unterstützen und die *Support Library* diese Funktion bereitstellt.

Listing 11.10 MapFragment im Layout

```
<fragment
    android:id="@+id/map_fragment"
    android:name="com.google.android.gms.maps.MapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentBottom="false"
    android:layout_alignParentTop="true"
    android:layout_marginBottom="50dp" />
```

LocationMapsActivity

Die LocationMapsActivity ist eine Activity, die ein Ergebnis zurückliefert. Durch Betätigung des OK-Buttons sendet die *LocationMapsActivity* ein Result-Intent an die aufrufende Activity. In der aufrufenden Activity wird das Ergebnis durch die überschriebene Methode *onActivityResult()* ausgewertet. In der Klasse *NoteTextActivity* ist die Funktion komplett abgebildet.

Listing 11.11 Die Klasse NoteTextActivity – eine Methode, die Result-Daten der Maps-Activity entgegennimmt

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
                               Intent data) {
    if(requestCode == LOCATION_ACTIVITY){
        closeKeyboard();
        String toastText = null;
        switch (resultCode){
            case Activity.RESULT_OK:
                if(data != null){
                    double latitude = data.getDoubleExtra("Latitude", 0);
                    double longitude = data.getDoubleExtra("Longitude", 0);
                    List<Address> address =
                        data.getParcelableArrayListExtra("Address");
                    Reminder locationReminder =
                        new Reminder(oldFile.getAbsolutePath(),
                                    fileHelper.deleteFileExtension(oldFile.getName()),
                                    NoteType.text);
                    locationReminder.setLatLon(latitude, longitude);
                    String street = null;
                    String zip_city = null;
                    String country = null;
                    if(address != null && address.size() == 1){
                        Address addr = address.get(0);
                        street = addr.getThoroughfare() != null ?
                            addr.getAddressLine(0) : "";
                        zip_city = addr.getPostalCode() + " " +

```

```

        addr.getLocality();
        country = addr.getCountryCode();
        locationReminder.setAddress(street);
        locationReminder.setSubAddress(zip_city + ";" + country);
    }
    locationReminder = saveReminder(locationReminder);
    toastText = street + '\n' + zip_city;
    Toast toast = Toast.makeText(getApplicationContext(),
        toastText, Toast.LENGTH_LONG);
    toast.show();
}
break;
case Activity.RESULT_CANCELED:
break;
default: break;
}//End switch-case
}
super.onActivityResult(requestCode, resultCode, data);
}

```

Der Start der *LocationMapsActivity* erfolgt in der *onReminderSelectionItemClick()*-Methode der *NoteTextActivity*. Im *case*-Zweig für den Erinnerungstyp *location* erfolgt zunächst die Überprüfung, ob mindestens ein Location-Provider zur Verfügung steht und ob eine Netzwerkverbindung existiert. Die Methoden zur Überprüfung werden durch die Klasse *LocationHelper* zur Verfügung gestellt. Sind beide Bedingungen erfüllt, wird die Methode *startLocationActivity()* zum Starten der *LocationMapsActivity* aufgerufen. Sind die beiden Bedingungen nicht erfüllt, erfolgt die Ausführung des *else*-Teils der *if*-Anweisung. Der Code im *else*-Teil erstellt einen *AlertDialog* und bringt diesen zur Anzeige. Der Alert-Dialog zeigt wieder eine Text-Message an, die in den String-Ressourcen definiert ist, ebenso der Titel. Die Verwendung des Alert-Dialogs haben wir schon erläutert.

Listing 11.12 Ausschnitt aus der *onReminderSelectionItemClick()*-Methode *NoteTextActivity*

```

case location:
    LocationHelper locHelper = new LocationHelper(this);
    if((locHelper.checkLocationService() >= 1) &&
        locHelper.checkNetworkIsOnline())){
        startLocationActivity();
    }else{
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle(getResources()
            .getString( R.string.location_alert_title));
        builder.setMessage(getResources()
            .getString(R.string.location_alert_message));
        builder.setPositiveButton(R.string.ok, null);
        AlertDialog alertDialog = builder.show();
        TextView messageText = (TextView) alertDialog
            .findViewById(android.R.id.message);
        messageText.setGravity(Gravity.CENTER);
        alertDialog.show();
    }
    break;
default: break;
} //onReminderSelectionItem

```

Die Methode `startLocationActivity()` erzeugt ein Intent, das im Kontext der `NoteTextActivity` ausgeführt wird, und legt die Klasse `LocationMapsActivity` als Empfänger fest. Mit `startActivityForResult()` wird das Intent versendet, und als zweiter Parameter wird eine Id zur Identifizierung des Intents gesendet. Die Id ist als statische Variable in der Klasse definiert.

Listing 11.13 Methode zum Starten der LocationMapsActivity

```
public void startLocationActivity() {
    Intent intent = new Intent(this, LocationMapsActivity.class);
    startActivityForResult(intent, LOCATION_ACTIVITY);
}
```

Die Prüfungen in der `NoteTextActivity` sind erfolgreich durchgeführt, und die `LocationMapsActivity` wird nun gestartet. In der `onCreate()` erfolgt lediglich die Referenzierung des Layouts und der Buttons sowie die Initialisierung der Klasse `LocationHelper` und `Geocoder`. Die Funktion von `LocationHelper` wurde schon beschrieben. Mit der Klasse `Geocoder` können Positionsdaten in Adressdaten decodiert werden, die Methode zum Decodieren liefert eine Liste des Typs `Address` zurück.

Listing 11.14 Methode `onCreate()` der LocationMapsActivity

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_location_maps);
    button_OK = (Button)findViewById(R.id.location_bottom_btn_ok);
    button_OK.setOnClickListener(this);
    button_Cancel = (Button)
        findViewById(R.id.location_bottom_btn_cancel);
    button_Cancel.setOnClickListener(this);
    mLocationHelper = new LocationHelper(this);
    geocoder = new Geocoder(this,
        getResources().getConfiguration().locale);
}
```

Die Erstellung der Karte erfolgt in der `onResume()`-Methode der `LocationMapsActivity`.

Zunächst wird die Map durch den `FragmentManager` referenziert. Die folgenden zwei Anweisungen registrieren den `onMapClickListener` und `onMarkerDragListener` in der Map.

Die folgende Anweisung erstellt eine Instanz von `GoogleMapOptions`. Mit dieser Klasse lässt sich die Map sehr umfangreich konfigurieren. Zunächst erfolgt die Festlegung des Kartentyps durch `.mapType(GoogleMap.MAP_TYPE_NORMAL)`. Es soll eine normale Karte angezeigt werden. Mit `.tiltGesturesEnabled(false)` wird die Funktion zum Schwenken/Neigen von Karten abgeschaltet, und `.compassEnabled(false)` deaktiviert den Kompass, der in der oberen linken Ecke angezeigt werden kann.

Die folgende `if`-Anweisung vergleicht den Rückgabewert der `LocationHelper`-Methode, und wenn die zurückgelieferte Position ungleich 0,0 ist, erfolgt die Zuweisung der Klassenvariable `position`. Die jetzt aufgerufene Methode `setMarkerStrings(position)` ermittelt durch Reverse-Geocodierung die Adresse der aktuelle Koordinaten, die der Methode als Parameter übergeben werden. Die Beschreibung der Methode erfolgt im weiteren Verlauf. Um die Adresse über den Markern anzeigen zu können, muss die Map den `InfoWindowAdapter`-Listener registrieren, das Interface `InfoWindowAdapter` ist in der Klasse implementiert.

Die folgenden Schritte definieren die Marker-Optionen des roten Positions-Markers, der am Ende der Methode der Map hinzugefügt wird. Zunächst erfolgt die Festlegung der Position in der Map, darauf folgt die Zuweisung von Titel und Snippet (zweite Zeile).

Mit `moveCamera(CameraUpdateFactory.newLatLng(position))` wird die Map auf die Koordinaten positioniert und `animateCamera(CameraUpdateFactory.zoomTo(13), 3000, null)` zoomt die Karte auf Level 13 mit einer Zeitverzögerung von 3.000 Millisekunden, das Interface `CancelableCallback`, als dritter Parameter, erhält `null`.

Zum Abschluss wird der Positions-Marker hinzugefügt und durch den Aufruf der Methode `showInfoWindow()` der Infotext angezeigt.

Listing 11.15 Erstellen der Map in der onResume()-Methode

```
@Override  
protected void onResume() {  
    super.onResume();  
    googleMap = ((MapFragment) getSupportFragmentManager()  
        .findFragmentById(R.id.map_fragment)).getMap();  
    googleMap.setOnMapClickListener(this);  
    googleMap.setOnMarkerDragListener(this);  
    GoogleMapOptions mapOptions = new GoogleMapOptions();  
    mapOptions.mapType(GoogleMap.MAP_TYPE_NORMAL)  
        .tiltGesturesEnabled(false)  
        .compassEnabled(false);  
    position = new LatLng(0, 0);  
    if (!mLocationHelper.getLatitude().equals(new LatLng(0,0))) {  
        position = mLocationHelper.getLatitude();  
    }  
    setMarkerStrings(position);  
    googleMap.setInfoWindowAdapter(this);  
    MarkerOptions positionMarker = new MarkerOptions()  
        .position(position)  
        .title(markerTitle)  
        .snippet(markerSnippet);  
    googleMap.moveCamera(CameraUpdateFactory.newLatLng(position));  
    googleMap.animateCamera(CameraUpdateFactory.zoomTo(12)  
        , 3000, null);  
    googleMap.addMarker(positionMarker).showInfoWindow();  
}
```

Wie schon beschrieben, soll ein eigener Marker der Map hinzugefügt werden und dieser bei Bedarf an eine frei wählbare Position in der Map verschoben werden. Damit Klick-Ereignisse der Map ausgewertet werden können, wird das Interface `OnMapClickListener` benötigt. Das Interface enthält nur eine abstrakte Methode `onClickMap()`, die überschrieben werden muss. Die `onClickMap()`-Methode erhält als Parameter den Punkt (Koordinaten) des Typs `LatLng`, auf den in der Map geklickt wurde.

Durch einen Klick auf die Map soll nur ein einziger Marker hinzugefügt werden. Die `if`-Anweisung überprüft, ob eine Instanz `reminderMarker` existiert. Ist dies nicht der Fall, erfolgt zunächst die Erstellung des `reminderMarkerOptions`-Objekts. Dem Objekt werden folgend die Eigenschaften zugewiesen, zunächst wird die Position in der Karte festgelegt. Darauf erfolgt die Festlegung, dass der Marker verschoben werden kann, durch `.draggable(true)`. Die Methode `setMarkerStrings()` ermittelt die Adresse aus der übergebenen Position, und im Folgenden werden die Eigenschaften `.title` und `.snippet` mit den ermittelten Strings festgelegt.

Die Form und Farbe des Marker Icons wird durch das Attribut `.icon` festgelegt. Der Marker erhält die Default-Form und die Farbe Blau, die durch die BitmapDescriptorFactory definiert werden.

Durch den Aufruf der Methode `.showInfoWindow()` erfolgt die Anzeige der Texte über den Marker.

Listing 11.16 Methode `onClickMap()` zum Hinzufügen eines neuen Markers

```
@Override
public void onClickMap(LatLng latLng) {
    if (reminderMarker == null) {
        MarkerOptions reminderMarkerOptions = new MarkerOptions()
            .position(latLng)
            .draggable(true);
        setMarkerStrings(latLng);
        reminderMarkerOptions.title(markerTitle);
        reminderMarkerOptions.snippet(markerSnippet);
        this.reminderMarker = googleMap.addMarker
            (reminderMarkerOptions
                .icon(BitmapDescriptorFactory
                    .defaultMarker(BitmapDescriptorFactory.HUE_BLUE)));
        this.reminderMarker.showInfoWindow();
    }
}
```

Das Interface `OnMarkerDragListener` erzwingt die Implementierung und das Überschreiben der abstrakten Methoden `onMarkerDrag()`, `onMarkerDragStart()` und `onMarkerDragEnd()`. Wir verwenden nur die Methode `onMarkerDragEnd()`, der Aufruf dieser Callback-Methode erfolgt, wenn das Verschieben des eigenen Markers beendet wurde.

Die Methode erhält beim Aufruf das Marker-Objekt, das verschoben wurde, als Parameter übergeben. Aus dem Objekt werden die Koordinaten durch `getPosition()` ermittelt. Im nächsten Schritt erfolgt die Ermittlung der Adresse zu der Position, und der Marker erhält durch `setTitle()` und `setSnippet()` die neuen Adressdaten. Durch den Aufruf der Methode `.showInfoWindow()` werden diese wieder angezeigt.

Listing 11.17 Methode `onMarkerDragEnd()`

```
@Override
public void onMarkerDragEnd(Marker marker) {

    LatLng position = new LatLng(marker.getPosition().latitude,
                                  marker.getPosition().longitude);
    setMarkerStrings(position);
    marker.setTitle(markerTitle);
    marker.setSnippet(markerSnippet);
    marker.showInfoWindow();
}
```

Die Methode `setMarkerStrings()` ermittelt durch Verwendung der Klasse `Geocoder` aus Positionsdaten die Adressdaten. Die Methode `.geocoder.getFromLocation()` der `Geocoder`-Klasse erhält als Parameter die Latitude, die Longitude der Position und die Anzahl der zu liefernden Ergebnisse. Das Ergebnis wird in der typisierten Liste `address` gespeichert. Bevor die Geocodierung ausgeführt wird, wird die Liste geleert, wenn diese Adressdaten enthält, und überprüft, ob der Geocoder initialisiert und verfügbar ist.



HINWEIS: Der Zugriff auf den Reverse Geocoding Service ist nicht immer gewährleistet. Schon seit Version 4.0 von Android treten immer wieder Probleme auf. Das Problem lässt sich meist durch einen Neustart des Telefons beheben.

Listing 11.18 Methode setMarkerStrings()

```
private void setMarkerStrings(LatLng latLng){  
    try {  
        if(address != null && address.size() > 0){  
            address.clear();  
        }  
        if(geocoder != null && Geocoder.isPresent()){  
            address = geocoder.getFromLocation(latLng.latitude,  
                                              latLng.longitude, 1);  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    if(address != null && address.size() > 0){  
        markerTitle = address.get(0).getAddressLine(1).toString();  
        markerSnippet = address.get(0).getAddressLine(0).toString();  
    } else {  
        markerTitle = getString(R.string.no_location);  
        markerSnippet = getString(R.string.no_address);  
    }  
}
```

Die *onClickView()*-Methode wird durch Betätigen der Dialog-Buttons OK und CANCEL aufgerufen. Werden die Buttons betätigt, soll die LocationMapsActivity beendet und das Result an die NoteTextActivity zurückgegeben werden. Hat der User die Erinnerungsposition durch einen neuen Marker (*reminderMarker*) festgelegt, soll diese Position und Adresse als Ergebnis zurückgegeben werden. Ist kein zusätzlicher Marker vorhanden, ist die ermittelte Position das Ergebnis.

Die Erstellung des Results erfolgt wieder durch ein Intent. Dieses sendet die Daten und den Result-Code. Da dieses ein Result-Intent ist, muss das Intent **final** sein. Die Ermittlung, welcher Button betätigt wurde, erfolgt, wie Ihnen schon bekannt ist, durch Auswertung der Id in der *switch/case*-Anweisung.

Im *case*-Zweig des OK-Buttons wird zunächst überprüft, ob in der Map ein *reminderMarker* vorhanden ist. Ist kein *reminderMarker*-Objekt auf der Map vorhanden, werden der Variablen *resultPosition* die beim Aufruf der Map ermittelten Positionsdaten zugewiesen.

Existiert ein *reminderMarker*-Objekt auf der Map, erfolgt die Ermittlung der *resultPosition* aus dem Marker-Objekt. Dem Intent werden die Daten mit der *putExtra*-Methode hinzugefügt, ebenso die Adresse. Da die Adresse eine typisierte Liste ist und somit mit dem Interface *Parcelable* („paketierbar“) erweitert werden kann, ist es möglich, dem Intent ein komplexeres Objekt als ein String-Objekt oder Integer-Objekt zu übergeben.

Mit der Methode *setResult()* erhält das *resultIntent* den Ergebnis-Code *RESULT_OK*, und mit *finish()* wird die *LocationMapsActivity* beendet.

Wurde der Button CANCEL betätigt, erhält das *resultIntent* nur den Ergebnis-Code *RESULT_CANCELED*, und keine weiteren Daten, und die Activity wird beendet.

Listing 11.19 Methode onClickView() zur Rückgabe des Results

```
@Override
    public void onClick(View v) {
        final Intent resultIntent = new Intent();
        switch (v.getId()) {
            case R.id.location_bottom_btn_ok:
                LatLng resultPosition;
                if(reminderMarker == null){
                    resultPosition = position;
                }
                else{
                    resultPosition = reminderMarker.getPosition();
                }
                resultIntent.putExtra("Latitude",
                        resultPosition.latitude);
                resultIntent.putExtra("Longitude",
                        resultPosition.longitude);
                resultIntent.putParcelableArrayListExtra("Address",
                        (ArrayList<? extends Parcelable>) address);
                setResult(RESULT_OK, resultIntent);
                finish();
                break;
            case R.id.location_bottom_btn_cancel:
                setResult(RESULT_CANCELED, resultIntent);
                finish();
                break;
            default:
                break;
        }
    }
}
```

Die Auswertung des Intents und die Speicherung der Daten in der Datenbank erfolgt in der Methode *onActivityResult()* der *NoteTextActivity*.

■ 11.4 Zwischenstand der App (Version 0.8)

Nachdem wir in Kapitel 10 bereits die Datenbank für die Speicherung von zeitbasierten und ortsbasierten Erinnerungen erstellt haben, zeigen wir Ihnen in diesem Kapitel, wie Sie unter Verwendung der Google Maps Api-Karten anzeigen, eigene Marker hinzufügen und durch Verwendung der Geocoder-Klasse aus Positionsdaten die Adressdaten ermitteln. Die so ermittelten Daten werden in der Datenbank gespeichert und in der Liste der MainActivity dargestellt.

In Kapitel 12 werden wir die Verwendung des AlarmManagers, ProximityAlerts und NotificationManagers erläutern, um den Anwender zeit- oder ortsbasiert zu informieren. In diesem Zusammenhang lernen Sie auch BroadcastReceiver kennen.

12

Zeit- und ortsbasierte Erinnerungen, lokale Notifications

Unsere App, die dem Erstellen und Verwalten von Notizen , inklusive Erinnerungen, dient, nähert sich langsam der Vollendung. In Kapitel 11 wurden folgende Funktionen vorgestellt:

- Karten mit einem Map-Fragment darstellen,
- Adressen für einen Ort mit Reverse Geocoding bestimmen und
- Code verzögert in einem eigenen Thread ausführen



Die Beispieldaten zu diesem Kapitel finden Sie unter www.downloads.hanser.de im Unterordner *scytenotes 0.9*.

In diesem Kapitel werden wir den grundlegenden Teil der App-Entwicklung abschließen, wobei folgende Funktionen umgesetzt werden:

- Erstellen von zeitbasiertem Alarm
- Erstellen von ortsbasiertem Alarm
- Empfangen von Broadcast-Intents
- Erstellen und Versenden von lokalen Nachrichten

■ 12.1 Zeitbasierte Erinnerungen mit dem AlarmManager

Beim Speichern einer Notiz entscheidet der User, ob eine Erinnerung erstellt werden soll und welcher Art diese ist. In diesem Abschnitt beschreiben wir die Erstellung einer zeitbasierten Erinnerung unter Verwendung der Klasse *android.app.AlarmManager*.

Für die Erstellung der zeit- oder ortsbasierten Erinnerungen werden die Methoden *setTimeReminderAlert()* und *setLocationReminderAlert()* der *ReminderAlertHelper*-Klasse aus dem Package *eu.scytenotes.data* verwendet.

Speichert der User die Notiz, so wird die Methode *saveReminder(Reminder reminder)* in der Activity aufgerufen. In der Methode erfolgt nach dem Speichern der Erinnerung in

der Datenbank der Aufruf `new ReminderAlertHelper(getApplicationContext(), reminder)`. Mit dem Aufruf des Konstruktors der Helper-Klasse erfolgt die Erstellung des Alarms, da dem Konstruktor als Parameter der *Context* und das *Reminder*-Objekt als initiale Parameter übergeben werden müssen.

Die Entscheidung, welcher Alarm erstellt werden muss, erfolgt schon durch Verwendung einer *switch/case*-Anweisung im Konstruktor (Listing 12.1). Die *switch/case*-Anweisung vergleicht den Typen des Reminders, und im entsprechenden *case*-Zweig wird die Methode zur Erstellung des Alarms aufgerufen. Am Beispiel der zeitbasierten Erinnerung wird die Methode `setTimeReminderAlert()` aufgerufen und erhält als Parameter die Referenz auf das *Reminder*-Objekt.

Listing 12.1 Konstruktor der *ReminderAlertHelper*-Klasse

```
public ReminderAlertHelper(Context context, Reminder reminder) {
    mContext = context;
    switch (reminder.getReminderType()) {
        case time:
            setTimeReminderAlert(reminder);
            break;
        case location:
            setLocationReminderAlert(reminder);
            break;
        default: break;
    }
}
```

In der Methode `setTimeReminderAlert()` wird als Erstes ein Intent (*alarmIntent*) erzeugt. Das *alarmIntent* wird als Content in ein Pending-Intent verpackt, das an den im *AlarmManager* registrierten Alarm angehängt wird.

Dem *alarmIntent* werden durch die Methode `.putExtra()` Informationen wie die Datenbank-ID des Reminders und der absolute Pfad zur Notiz hinzugefügt. Um einen Alarm erstellen zu können, muss als Erstes eine Referenz auf den *AlarmManager* erzeugt werden, dass das Objekt *alarmManager* initialisiert werden soll. Mit *alarmManager.set(...)* wird der eigentliche Alarm erstellt. Der erste Parameter legt fest, welche Art von Alarm erstellt werden soll. Neben dem von uns verwendeten Typ zu einem festen Zeitpunkt können unter anderem Intervalle erstellt werden. Der zweite Parameter legt den Zeitpunkt des Alarms in Millisekunden fest. Als letzter Parameter wird das Intent angegeben, das beim Auslösen des Alarms versendet werden soll.

Listing 12.2 Erstellen eines Alarms im *AlarmManager*

```
private void setTimeReminderAlert(Reminder reminder){
    Intent alarmIntent =
        new Intent(this.mContext,ReminderReceiver.class);
    int reminderID = (int) reminder.getDataBaseId();
    alarmIntent.putExtra("reminderId", reminderID);
    alarmIntent.putExtra("reminderNote", reminder.getFilePath());
    alarmIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);
    PendingIntent alarmPendingIntent =
        PendingIntent.getBroadcast(mContext,
            reminderID, alarmIntent, 0);
    AlarmManager alarmManager = (AlarmManager)
```

```
mContext.getSystemService(Context.ALARM_SERVICE);
alarmManager.set(AlarmManager.RTC_WAKEUP,
    reminder.getDateTime(),
    alarmPendingIntent);
}
```

Damit der Alarm eingerichtet werden kann, benötigt die App die Berechtigung zum Erstellen von *Alarm*. Vergeben Sie im Android-Manifest die Berechtigung `com.android.alarm.permission.SET_ALARM`.

Listing 12.3 Permission SET_ALARM im Android-Manifest

```
<uses-permission android:name="com.android.alarm.permission.SET_ALARM"/>
```

■ 12.2 BroadcastReceiver und NotificationManager

Im vorherigen Abschnitt haben Sie den zeitbasierten Alarm eingerichtet. Damit die vom `AlarmManager` versendeten Broadcast-Intents empfangen und die Notifications erstellt werden können, wird ein `BroadcastReceiver` benötigt, der diese Broadcasts entgegennimmt und die übermittelten Informationen weiterverarbeitet.

Diese Aufgabe übernimmt die Klasse `ReminderReceiver` aus dem Package `eu.scyte.notes.data`. Die Klasse `ReminderReceiver` erbt von der Klasse `BroadcastReceiver`. Damit dieser Receiver verwendet werden kann, muss dieser im Android-Manifest registriert werden, und es muss ein weiterer Intent-Filter erstellt werden.

Listing 12.4 Intent-Filter des Android-Manifests

```
<intent-filter>
    <action android:name="android.intent.action.SET_ALARM" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Die `onReceive()`-Methode wird bei Eintreffen eines Broadcast-Intents aufgerufen und nimmt das Intent entgegen. Die vom Intent übermittelten Daten des Bundle werden durch die `.getExtra()`-Methode im Bundle-Objekt zwischengespeichert und dann durch Zugriff mit dem Key aus dem Bundle extrahiert. Um Notifications versenden zu können, wird ein neues Intent benötigt. Dieses Intent erhält als Empfänger die `MainActivity` der App.

Mit dem `NotificationManager` wird die Notification, die in der `NotificationBar` des Telefons angezeigt werden soll, erstellt. Die Notification wird nun weiter konfiguriert. Es wird der Titel und der anzuzeigende Text festgelegt, die Meldung, die als Ticker beim Eintreffen angezeigt werden soll, und das anzuzeigende Icon. Mit dem Notification-Flag `FLAG_AUTO_CANCEL` wird sichergestellt, dass bei einem Klick auf die Nachricht nur eine Instanz der App gestartet ist und gegebenenfalls eine offene Activity geschlossen wird. Erst durch Aufruf der `.notify()`-Methode wird die Notification ausgeführt, der ebenfalls ein Intent angehängt wird.

Zum Abschluss erfolgt die Anzeige einer kurzen Bildschirmmeldung, die durch einen Toast erzeugt wird.

Listing 12.5 Methode onReceive() des ReminderReceivers

```
@Override
public void onReceive(Context context, Intent intent) {
    try {
        Bundle bundle = intent.getExtras();
        String message = bundle.getString("alarm_message");
        String reminderPath = bundle.getString("reminderNote");
        Intent notifyIntent = new Intent();
        notifyIntent.setClass(context, MainActivity.class);
        PendingIntent pendingIntent = PendingIntent
            .getActivity(context, 0, notifyIntent,
        android.content.Intent.FLAG_ACTIVITY_TASK_ON_HOME);
        notificationManager = (NotificationManager) context
            .getSystemService(Context.NOTIFICATION_SERVICE);
        Notification notification = new Notification.Builder(context)
            .setContentTitle("ContentTitle")
            .setContentText("ContentText")
            .setSmallIcon(R.drawable.ic_notification_overlay)
            .setContentInfo("ContentInfo")
            .setTicker("ticker")
            .setContentIntent(pendingIntent)
            .setDefaults(DEFAULT_ALL)
            .getNotification(); //ab Api-Level 16 .build
        notification.flags |= Notification.FLAG_AUTO_CANCEL;
        notificationManager.notify(0, notification);
        Toast.makeText(context, message, Toast.LENGTH_LONG).show();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

■ 12.3 Ortsbasierte Erinnerungen mit ProximityAlerts

Um eine ortsbasierte Erinnerung zu erstellen, gibt zwei Möglichkeiten. Zum Einen können Sie einen eigenen Service implementieren, der auf Änderungen der Position durch den Location-Listener informiert wird und anhand von Kriterien einen Broadcast sendet. Die zweite Möglichkeit ist, den ProximityAlert der Klasse *LocationManager* zu nutzen. Beide Möglichkeiten haben ihre Vor- und Nachteile. Wollen Sie einen eigenen Service nutzen, müssen Sie nicht nur auf Änderungen der Position reagieren und die Logik der Annäherung implementieren, sondern auch Änderungen an Einstellungen der Standortdienste, Provider aktivieren oder deaktivieren, überwachen und darauf reagieren. Wollen Sie dagegen den ProximityAlert nutzen, ist die Verwendung genauso einfach wie die Erstellung eines Alarms, jedoch ist die Genauigkeit nicht so gut wie bei einem Service, der auf Änderungen der Position reagiert.

Wir haben uns für die Nutzung der ProximityAlerts entschieden. Für die Nutzung der ProximityAlerts spricht, dass diese einfach zu implementieren sind, kein Management für die Aktivierung und Deaktivierung von Sensoren erfolgen muss, und diese keine Probleme mit fehlerhaften Geräten verursachen. Gegen die ProximityAlerts spricht, dass diese erst ab einem eingestellten Radius von > 1.000 Metern zuverlässig mit dem Provider-Network arbeiten. Der Network-Provider errechnet aus den Standortdaten der Sendemasten die Position des Gerätes. Somit sind Radien kleiner < 1000 m nicht möglich, um Alarmierungen auszulösen.

In Abschnitt 12.1 haben wir die Erstellung des zeitbasierten Alarms ausführlich erläutert. Die Erstellung des ortsbasierten Alarms erfolgt auf die gleiche Weise. Allerdings wollen wir kurz die Verwendung der `addProximityAlert()`-Methode der Klasse `LocationManager` erläutern.

Die `addProximityAlert()`-Methode des Location-Managers legt die Position durch Angabe der Koordinaten des Reminders fest, an welche das `proximityIntent` versendet werden soll. Mit dem Radius wird festgelegt, in welchem Umkreis der Alarm ausgelöst werden soll. Mit `-1` wird der festgelegt, dass der Alarm nicht verfällt. Durch Angabe einer Zeitspanne in Millisekunden könnte festgelegt werden, nach welcher Dauer der Alarm verfällt. Mit dem letzten Parameter wird wieder das Intent angehängt, das beim Auslösen des Alarms versendet werden soll.

Listing 12.6 Methode, um ortsbasierten Alarm zu erzeugen

■ 12.4 Zwischenstand der App (Version 0.9)

Nun ist die eigentliche Programmierung der Beispiel-App vollständig umgesetzt. Die App ist voll funktionsfähig. Anhand der App-Erstellung wurden zahlreiche Themen der Android-Entwicklung vorgestellt: Erstellung von Notizen in Form von Text, Bild und Audio sowie die Möglichkeit, zeit- und ortsbasierte Erinnerungen zu erstellen.

In Kapitel 13 werden wir uns abschließend noch einigen Feinheiten widmen, bevor wir in Kapitel 14 die App im Google Play Store veröffentlichen.

13

Lokalisierung, Icons und Startbilder – Vorbereitung für die Veröffentlichung im Play Store

In den Kapiteln 4 bis 12 haben wir eine vollständige App entwickelt, mit der man Notizen und Erinnerungen erstellen kann. Dabei wurden viele Themen der Entwicklung unter Android berücksichtigt, die in der einen oder anderen Form immer wieder in der App-Entwicklung vorkommen.



Die Beispieldaten zu diesem Kapitel finden Sie unter www.downloads.hanser.de im Unterordner *scytenotes 1.0*.

In diesem Kapitel bereiten wir die App für die Veröffentlichung im Play Store vor und verleihen ihr den letzten Schliff. Weiterhin überprüfen Sie den Code mit Android Lint.

■ 13.1 Die App lokalisieren

Die Lokalisierung der App besteht im Wesentlichen darin, die verwendeten String-Ressourcen für die einzelnen unterstützten Sprachen bereitzustellen. Sie sollten jedoch beachten, dass es je nach Umfang und Funktionalität der App nicht ausreicht, diese nur mit einer weiteren Sprache zu unterstützen. Neben den offensichtlichen Unterschieden, wie der Verwendung von unterschiedlichen Maßeinheiten (metrisch in Europa und Zoll in den USA und England), sind auch kulturelle Eigenschaften für den Erfolg in den Regionen von Bedeutung.

Google hat vor einiger Zeit die Lokalisierung von Apps für verschiedene Regionen und Sprachen untersucht und stellt mittlerweile eine Checkliste zur Lokalisierung zur Verfügung. Diese finden Sie unter <http://developer.android.com/distribute/googleplay/publish/localizing.html>.

13.1.1 String-Ressourcen lokalisieren

Wie schon oft im Verlauf des Buchs erwähnt, ist die Verwaltung von unterschiedlichen Ressourcen, zu denen auch die Strings zählen, sehr effizient umgesetzt und die Bereitstellung von Ressourcen für unterschiedliche Sprachen sehr einfach.

Um String-Ressourcen für eine weitere Sprache hinzuzufügen, gehen Sie wie folgt vor.

1. Markieren Sie als Erstes den Ordner `/res/values` und wählen im Kontextmenü über Rechtsklick **NEW** und dann **ANDROID XML FILE** aus.
2. Im folgenden Dialog vergeben Sie als Namen `strings` und gehen mit **NEXT** weiter. Achten Sie darauf, dass der *Resource Type* = `values` ist.
3. Wählen Sie als Qualifizierer `LANGUAGE` und den Sprachcode `de`. Speichern Sie die Datei. Es wurde nun ein neues Verzeichnis `/res/values-de` mit einer leeren Datei `strings.xml` angelegt.
4. Kopieren Sie den Inhalt der Standard-String-Ressource in die neu angelegte Datei und übersetzen nun den Inhalt.

Es müssen nicht alle String-Elemente in der sprachspezifischen Ressource vorhanden sein. Ist ein String-Element nicht vorhanden, wird der String aus der Default-String-Ressource verwendet.

■ 13.2 Icons und Bilder

Neben den in der App selbst verwendeten Icons, benötigen Sie noch weitere Icons und Bilder für die Veröffentlichung im Play Store.

Bis jetzt besitzt die Beispiel-App noch kein Launcher Icon. Launcher Icons müssen eine Größe von 48×48 dp haben. Die einfachste Möglichkeit, aus einem Image ein passendes Icon zu erstellen, ist die Nutzung des *Android Asset Studios*, welches Sie unter <http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html> finden. Nutzen Sie den Google Chrome Browser, wenn Sie mit dem Asset Studio arbeiten, unter Firefox oder Internet Explorer sind nicht alle Funktionen verfügbar.

Wählen Sie zur Erstellung den Link **LAUNCHER ICONS**. Sie können nun im Asset Studio die Bildvorlage hochladen und das vom Asset Studio erstellte Paket des Launcher-Icons für die unterschiedlichen Display-Typen downloaden. Nach erfolgreichem Download entpacken Sie die zip-Datei und kopieren die einzelnen Icon-Dateien in die entsprechenden Ordner `/res/drawable-hdpi` etc. in Eclipse. Beim nächsten Start aus Eclipse heraus erscheint nun das Icon in der App.

Um die App im Play Store veröffentlichen zu können, benötigen Sie jedoch noch weitere Bilder. Google stellt hier ein paar Mindestanforderung, die erfüllt sein müssen, damit die App im Play Store angezeigt wird.

Sie benötigen

- für ein Preview der App mindestens 2 Screenshots Ihrer App mit mindestens 320 px Seitenlänge sowie
- ein hochauflösendes Icon mit 512×512 px des Typs *png* und 32 bit (mit Alpha), das im Play Store angezeigt wird.

■ 13.3 Der letzte Feinschliff

Wir wollen dem User noch ein bisschen Komfort zum Löschen von Notizen gönnen. In der Start-Activity kann er bis jetzt nur neue Notizen hinzufügen oder bestehende durch einen Klick öffnen. Zusätzlich soll der User durch einen Long-Klick auf eine Notiz einen Auswahlmodus starten, um mit einem einfachen Klick weitere Notizen zu selektieren, um diese anschließend löschen zu können.

Nach einem langen Klick auf ein Notiz-Item und Auswahl weiterer Notizen stellt sich die Oberfläche wie in Bild 13.1 dar.

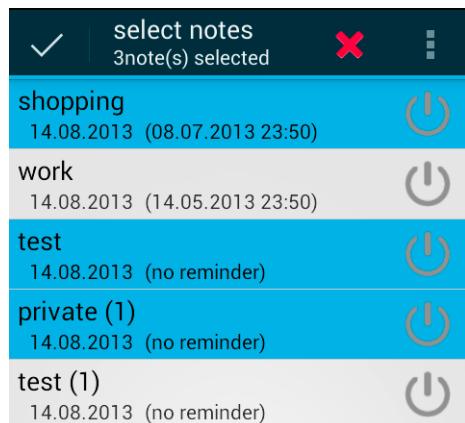


Bild 13.1

ListView mit selektierten Notizen

Diese Funktion stellt die innere Klasse *ListViewCallback* aus Listing 13.1 der *MainActivity* bereit. Erstellen Sie die innere Klasse aus dem Listing in der Klasse *MainActivity*.

Die Klasse implementiert das Interface *MultiChoiceModeListener*, das eine Mehrfachselektion von Items der Notizen-*ListView* ermöglicht. Durch den Long-Klick auf eine Notiz in der Liste wird die *onCreateActionMode()*-Methode aufgerufen. In dieser wird zunächst das Menü der *ActionBar* neu erstellt, und der Titel erhält einen neuen Wert. Die Verwendung des *ActionMode*-Menüs entspricht dem der *ActionBar*. Die Methode für die Nutzung der Menü-Einträge ist *onActionItemClicked()*. In der *switch/case*-Anweisung der Methode erfolgt wieder die Ermittlung, welches Menü-Item betätigt wurde, durch die Id des Menü-Items. Beim Betätigen des Delete-Items erfolgt der Aufruf der *deleteSelectedNotes()*-Methode, die die ausgewählten Notizen löscht und den *ActionMode* durch *mode.finish()* beendet.

Die Methode *onDestroyActionMode()* wird beim Verlassen des *ActionMode* aufgerufen.

Die Ausführung der Methode `onItemCheckedStateChanged()` erfolgt immer, wenn eine Notiz in der `ListView` selektiert wird und das Attribut `checked` des Items den Wert `true` erhält. In der Methode speichert die Integer Variable `checkedNotes` die Anzahl der selektierten Notizen, die durch die Methode `onItemCheckedStateChanged()` ermittelt wird, und bringt diese im Untertitel der `ActionBar` mit `mode.setSubtitle(...)` zur Anzeige.

Listing 13.1 Die innere Klasse `ListViewCallback` zur Selektion von Notizen

```
private class ListViewCallback
    implements ListView.MultiChoiceModeListener {

    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main, menu);
        mode.setTitle(getString(R.string.select_notes));
        return true;
    }

    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false;
    }

    @Override
    public boolean onActionItemClicked(ActionMode mode,
                                       MenuItem item) {
        switch (item.getItemId()) {
            case R.id.main_menu_delete:
                deleteSelectedNotes();
                mode.finish();
                return true;
            default: return false;
        }
    }

    public void onDestroyActionMode(ActionMode mode) {
    }

    public void onItemCheckedStateChanged(ActionMode mode,
                                         int position,
                                         long id,
                                         boolean checked) {
        final int checkedCount =
            noteListView.getCheckedItemCount();
        mode.setSubtitle(checkedCount + " "
                        + getString(R.string.notes_selected));
    }
}
```

Ergänzen Sie die `onResume()`-Methode der `MainActivity` mit den beiden Code-Zeilen aus Listing 13.2. Die erste Zeile legt den Auswahlmodus der `ListView` fest. In der zweiten Zeile erfolgt die Festlegung des Auswahl-Listeners. Da die innere Klasse `ListViewCallback` das Interface `MultiChoiceModeListener` implementiert, erfolgt die Zuweisung implizit durch Initialisierung der Klasse.

Listing 13.2 Auszug aus der onResume()-Methode

```
noteListView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
noteListView.setMultiChoiceModeListener(new ListViewCallback());
}
```

Bei der Selektion einer Notiz der ListView wird der Wert der Eigenschaft *checked* des ListView Items auf *true* geändert.

Der Aufruf der Methode zum Löschen der selektierten Notizeinträge (*deleteSelectedNotes()*) erfolgt in der überschriebenen Methode *onOptionsItemSelected(MenuItem item)* durch Auswahl des Menüeintrages *delete* der ActionBar.

Die Iteration mit einer *for*-Schleife über den *ListAdapter* und die Prüfung des Attributs *checked* auf **true** ist sehr langsam. Android bietet mit dem *SparseBooleanArray* einen Mechanismus, um Integer Keys zu booleschen Werten abzubilden. Hierbei kann es im Gegensatz zu normalen Arrays zu Lücken im Index kommen. Die Verwendung dieses speziellen Arrays ist wesentlich performanter und resourcenschonender als die Verwendung einer *HashMap*.

Der spezielle Array-Typ *SparseBooleanArray* aus dem Paket *android.util* bildet den *Key* jedes Items ab, dessen Wert *checked=true* ist, und speichert diesen.

In der *for*-Schleife wird das Array durchlaufen. Der Key der zu löschenen Items wird ausgelesen und ein Notiz-Objekt wird erzeugt. In der *if*-Anweisung wird überprüft, ob der Löschvorgang der Datei durch die *FileHelper* Methode *deleteFile(File)* erfolgreich war. Ist die Bedingung erfüllt, wird der Notizeintrag mit *listAdapter.remove(note)* aus der Liste entfernt. Da der Index des *SparseBooleanArray* Lücken aufweisen kann, muss die *for*-Schleife das Array rückwärts durchlaufen.

Listing 13.3 Methode zum Löschen ausgewählter Notizen

```
private void deleteSelectedNotes() {
    SparseBooleanArray checkedNotes =
        noteListView.getCheckedItemPositions();
    FileHelper fileHelper = new FileHelper();
    try {
        for (int i = (checkedNotes.size() - 1); i >= 0; i--) {
            Note note = (Note)
                listAdapter.getItem(checkedNotes.keyAt(i));
            if (fileHelper.deleteFile(note.getFile()) == true) {
                listAdapter.remove(note);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Zum jetzigen Zeitpunkt können mehrere Notizen selektiert und gelöscht werden. Jedoch kann der User nicht erkennen, welche Notizen er zum Löschen selektiert hat. Dies ändern Sie recht schnell durch die Verwendung eines Styles für das Layout des ListItems.

Erstellen Sie zunächst einen neuen Style (Listing 13.4) in der Ressource *styles.xml* und weisen dem Layout *listitem_note.xml* den Eintrag *style="@style/listItem"* im Layout-Tag zu. Jetzt müssen Sie nur noch die Ressource *selector.xml* (Listing 13.5) im Ordner */res/drawable* erstellen.

Wird eine Notiz ausgewählt, ist der Status des ListItems *activated*. Im Selector ist die Hintergrundfarbe einer View, deren Status *activated* ist, Blau. Durch Zuweisung des Styles im Layout des ListItems wird diese Eigenschaft festgelegt.

Listing 13.4 Style des ListItems

```
<style name="listItem">
    <item name="android:background">@drawable/selector</item>
</style>
```

Listing 13.5 Ressource selector.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android=
            "http://schemas.android.com/apk/res/android">
    <item android:state_activated="true"
          android:drawable="@android:color/holo_blue_light" />
</selector>
```

Neben den schon genannten Punkten der String-Lokalisierung und Verwendung von Icons sollten Sie nicht vergessen, auch die Hilfe- oder Info-Dateien der App zu lokalisieren.

Wir beschränken uns in der App auf die Lokalisierung der Hilfe-Datei und bieten diese zusätzlich in deutscher Sprache an. Sie haben die englische Hilfe-Datei *help.html* aus dem Ordner */res/raw* ins Deutsche übersetzt und müssen nur noch den Ressourcen-Ordner für die deutsche Sprache anlegen und die Datei in diesem Ordner speichern. Legen Sie dazu im Verzeichnis */res* den Ordner */res/raw-de* an und speichern die deutsche Hilfe-Datei darin ab.

13.3.1 Mit Android Lint den Code verbessern

Im Laufe der Entwicklung sind Sie immer wieder auf Warnungen oder Fehler in der Problem-View hingewiesen worden. Warnungen sind grundsätzlich nicht tragisch, besonders wenn diese auf Probleme mit im Code festgelegten Strings oder auf fehlende Kontextbeschreibungen hinweisen. Wenn Sie sich sicher sind, dass Sie diese nicht benötigen oder Texte im Code überschreiben, können Sie diese natürlich ignorieren.

Mit *Android Lint* überprüfen Sie die Qualität Ihres Codes speziell für Android. *Android Lint* weist Sie auf die gängigsten Fehler und Probleme, die speziell in der Android-Entwicklung auftreten können, hin. *Android Lint* wird auch automatisch beim Export der App ausgeführt. Die Ergebnisse der Überprüfung mit *Android Lint* werden in der View *Lint Warnings* angezeigt.

In der Standardkonfiguration von *Android Lint* werden auch nicht lokalisierte Strings als Fehler angezeigt. Diese Einstellung muss verändert werden, da auch die Bibliothek der Google Play Services überprüft wird und der Export des Projekts sonst scheitern würde.

Öffnen Sie hierzu die Einstellungen von Eclipse (Preferences) und navigieren Sie zum Eintrag *Lint Error Checking* im Zweig *Android*. Ändern Sie den Schweregrad (Severity) der Meldung mit der Id = *Missing Translation* auf Warnung, wie in Bild 13.2 zu sehen.

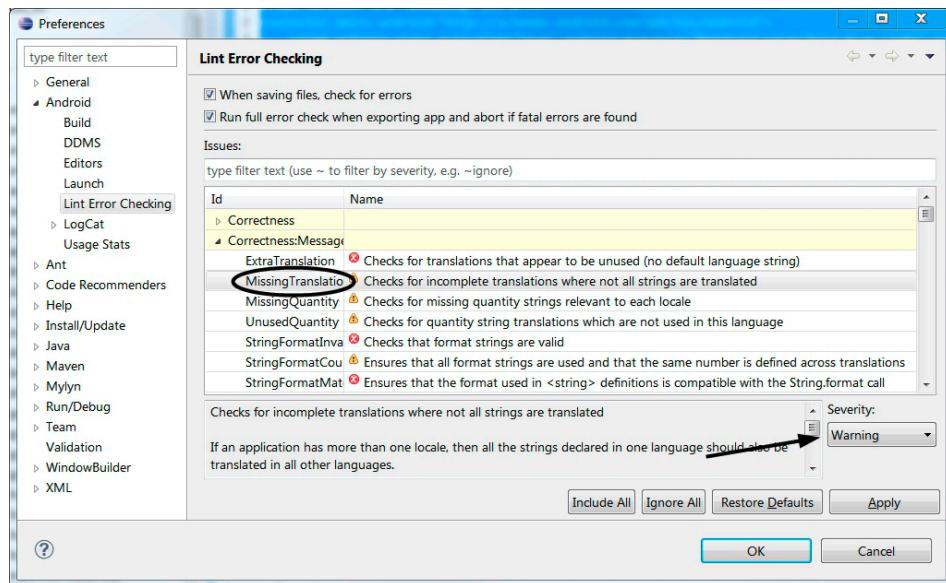


Bild 13.2 Einstellung „Android Lint“ in Eclipse

14

Veröffentlichung einer App im Play Store

Sie haben Schritt für Schritt eine umfangreiche App zur Erstellung von Notizen, mit der Option, sich an diese erinnern zu lassen, entwickelt. Es ist nun an der Zeit, diese im Google Play Store zu veröffentlichen. In Kapitel 13 haben Sie schon alle Vorbereitungen getroffen, damit die App schnell veröffentlicht werden kann.

In diesem Kapitel stehen noch folgende Punkte an:

- Export der App mit neuer Signatur
- Veröffentlichung der App im Google Play Store

■ 14.1 App mit eigener Signatur exportieren

Bevor Sie die App im Play Store bereitstellen können, müssen Sie diese als Paket exportieren, um sie später in den Play Store hochladen zu können. Google verlangt im Play Store signierte apk-Pakete, und Sie sollten hierfür nicht das bisher verwendete Debug-Zertifikat verwenden, sondern ein neues Zertifikat erstellen. Sie müssen jetzt nicht mit dem *Java keytool* ein neues Zertifikat erstellen, dies kann in einem Schritt beim Export erledigt werden.

Den Export Ihrer App starten Sie mit den Android-Tools. Markieren Sie dazu Ihr Projekt in Eclipse und rufen im Kontextmenü über Rechtsklick die ANDROID TOOLS auf und wählen den Export SIGNED APPLICATION PACKAGE ..., wie in Bild 14.1 zu sehen.

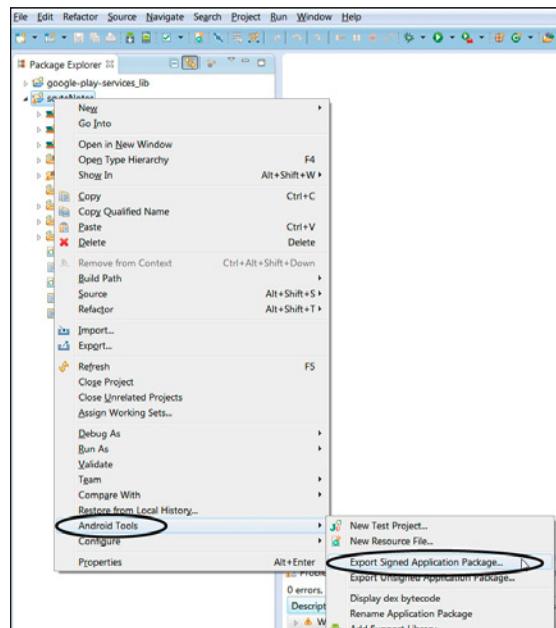


Bild 14.1 Export der App mit Android Tools

Bei Ausführung des Export-Tools wird automatisch Android Lint ausgeführt. Sollten Fehler bei der Überprüfung auftreten, wird dies im Dialog angezeigt. Wie Sie jedoch in Bild 14.2 erkennen können, ist alles in Ordnung, denn Sie haben ja schon vorher alles überprüft.

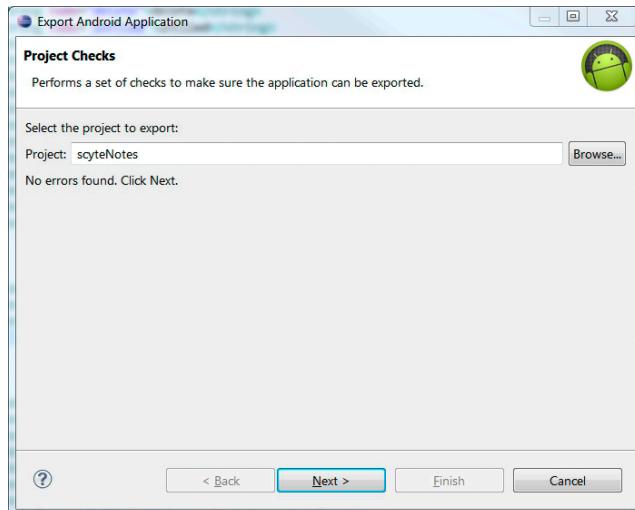


Bild 14.2 Export des Projekts und Ergebnis der Überprüfung durch Android Lint

Im nächsten Schritt erstellen Sie aus Eclipse heraus einen neuen Keystore und ein neues Zertifikat für die Signierung der App. Das Zertifikat sollten Sie sicher aufbewahren und sich das Passwort merken, um eventuell spätere Versionen wieder signieren zu können (Bild 14.3).

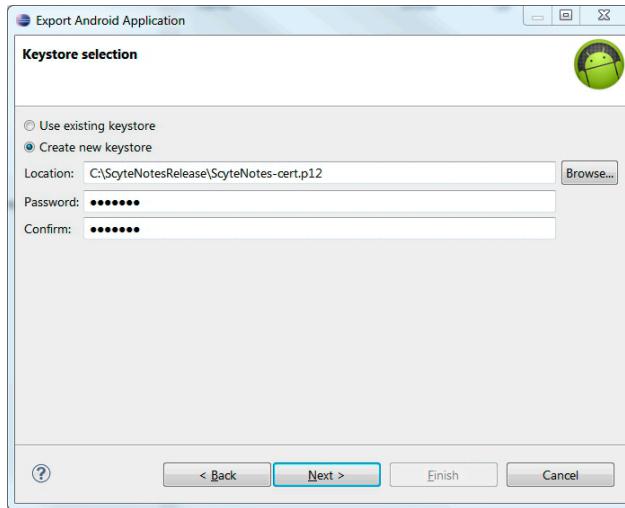


Bild 14.3 Neuen Keystore mit Passwort erstellen

Jetzt geben Sie die Informationen zum Zertifikat ein (Bild 14.4).

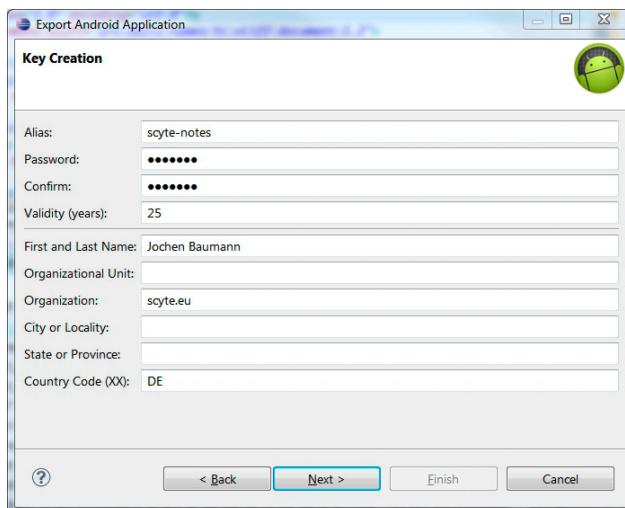


Bild 14.4 Key und Angaben des Zertifikats

Im letzten Schritt schließen Sie den Export ab (Bild 14.5). Im Anschluss können Sie die apk-Datei direkt in den Google Play Store hochladen und die App veröffentlichen.

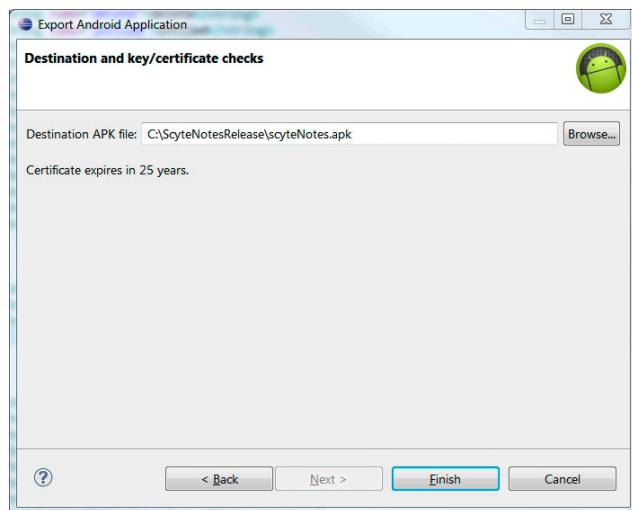


Bild 14.5 Export der App abschließen

■ 14.2 App in den Play Store hochladen

Zunächst müssen Sie sich natürlich im Google Play Store anmelden. Die Adresse für die Anmeldung als Entwickler lautet: <https://play.google.com/apps/publish>.

Nachdem Sie sich im Play Store angemeldet haben, können Sie direkt Ihre zuvor exportierte apk-Datei hochladen. Dabei wird automatisch ein neues App-Projekt im Play Store erstellt. Da wir die App bei der Entwicklung per default in Englisch erstellt haben, müssen wir natürlich bei der Anlage des Projekts als Standardsprache Englisch auswählen, und in unserem Fall US-Englisch, wie in Bild 14.6 zu sehen.

A screenshot of the 'NEUE APP HINZUFÜGEN' (New App Add) form. The title is 'NEUE APP HINZUFÜGEN'. It has a 'Standardsprache *' dropdown set to 'Englisch (Vereinigte Staaten) – en-US'. There is a 'Titel *' input field with the value 'scyte notes'. Below it is a note '11 von 30 Zeichen'. Under 'Womit möchten Sie beginnen?', there are three buttons: 'APK hochladen' (highlighted in blue), 'Store-Eintrag vorbereiten', and 'Abbrechen'.

Bild 14.6 Neue App, neues Projekt im Play Store hinzufügen

Die Funktionen, Möglichkeiten und die Lokalisierung im Play Store sind mittlerweile sehr weit fortgeschritten und bieten auch eine sehr gute Bedienung in deutscher Sprache. Sie können Ihre App derzeit sofort veröffentlichen oder sogar Testversionen einer App bereitstellen.

Wenn Sie eine App im Ordner ALPHA-TEST oder BETA-TEST bereitstellen, so wird diese nicht im Play Store als App auf den Geräten erscheinen. Sie können nur den Link im Play Store über eine Gruppe in Google Groups oder Google+ verteilen. Sie finden weitere Informationen unter <https://support.google.com/googleplay/android-developer/answer/3131213?hl=de>.

Nachdem Sie nun die App in den Play Store hochgeladen haben, können Sie sich eine Zusammenfassung durch Klick auf den App-Namen anzeigen lassen. Anhand der verwendeten Funktionen wird die App im Play Store der Geräte angezeigt, die diese unterstützen. Sollte Ihnen bekannt sein, dass es bei bestimmten Geräten Problem gibt, so können Sie diese schon vor der Veröffentlichung ausschließen.

The screenshot displays a detailed view of an APK file's metadata. At the top, it says 'DETAILS ZUR APK-DATEI' (Details of the APK file) for the package 'eu.scytle.notes'. Below this, there is a table with the following data:

eu.scytle.notes		
Versionscode 10	Name der Version 1.0	Größe 0,66 MB
Unterstützte Android-Geräte 1733 Geräte		
API-Ebenen 14+		
Bildschirm-Layouts 4 Bildschirmlayouts		
Lokalisierungen Standard + 49 Sprachen		
Funktionen 5 Funktionen		
Erforderliche Berechtigungen 12 Berechtigungen		
OpenGL ES-Versionen 2.0+		
OpenGL-Texturen alle Texturen		
Hochgeladen am 11.08.2013 15:40:11		
SHA1-Funktion der APK-Datei f971c061cf7fd6df6c9aa4619459cecdfd670615		

At the bottom left is a 'Schließen' (Close) button.

Bild 14.7 Übersicht zur App im Play Store

Wollen Sie sich die Geräte, die nicht unterstützt werden, anzeigen lassen, so öffnen Sie die Liste *unterstützte Geräte* und wählen in der Auswahl *nicht unterstützte* Geräte aus.

■ 14.3 Store-Eintrag erstellen und Icons und Screenshots verwalten

Sie sollten, wenn es Ihre App erfordert, für jede Lokalisierung eigene Screenshots erstellen. Das fördert die Akzeptanz bei den Interessenten im Play Store. Außerdem ist es günstig, den Beschreibungstext der App so zu gestalten, dass dem zukünftigen Nutzer klar ist, weshalb Ihre Anwendung bestimmte Rechte benötigt. In letzter Zeit ist zu beobachten, dass die Benutzer immer kritischer die Anforderungen von Rechten wie Internet, Location oder Zugriff auf Adressen betrachten. Ist Ihr Beschreibungstext so erstellt, dass der Nutzer die Anforderung benötigter Rechte auf die Funktionalitäten der App zurückführen kann, so wird diese auch problemlos akzeptiert.

Sie sollten den Aufwand nicht scheuen, für jede unterstützte Sprache, einen eigenen Beschreibungstext zu verfassen.

■ 14.4 Preisgestaltung und Vertrieb

Sie können Ihre Apps kostenpflichtig oder kostenlos im Google Play Store veröffentlichen. Um eine App kostenpflichtig veröffentlichen zu können, müssen Sie Verkäuferinformationen derzeit in Google Checkout und ab 20. November 2013 in Google Wallet angeben.



HINWEIS: Haben Sie eine App kostenlos veröffentlicht, so kann dies später nicht mehr geändert werden.

Obwohl wir die Beispiel-App kostenlos anbieten, wollen wir noch kurz auf die Preisgestaltung von kostenpflichtigen Angeboten eingehen.

Sie können die Länder, in denen die App angeboten werden soll, einschränken. Dies gilt für kostenpflichtige wie auch für kostenlose Angebote. Außerdem besteht die Möglichkeit, in einigen Ländern das Angebot auf Mobilfunkanbieter einzuschränken.

Bieten Sie eine App kostenpflichtig an, so sollten Sie die Preisgestaltung genauer betrachten. Sie bietet die Möglichkeit, für einige Länder einen abweichenden Preis festzulegen. Sie können natürlich den Preis automatisch für alle Länder übernehmen und die Landeswährungen umrechnen lassen, jedoch sollten Sie die regionalen Bedingungen beachten, auch im Euro-Raum. Ein Preis, der in Deutschland als akzeptabel gilt, muss in Österreich oder Spanien nicht akzeptiert werden, oder vielleicht können Sie in der Schweiz einen höheren Preis erzielen und sollten für Ungarn und Indien diesen senken.

Sie können den Preis für jedes Land einzeln angeben und so auf die regionalen Unterschiede eingehen.

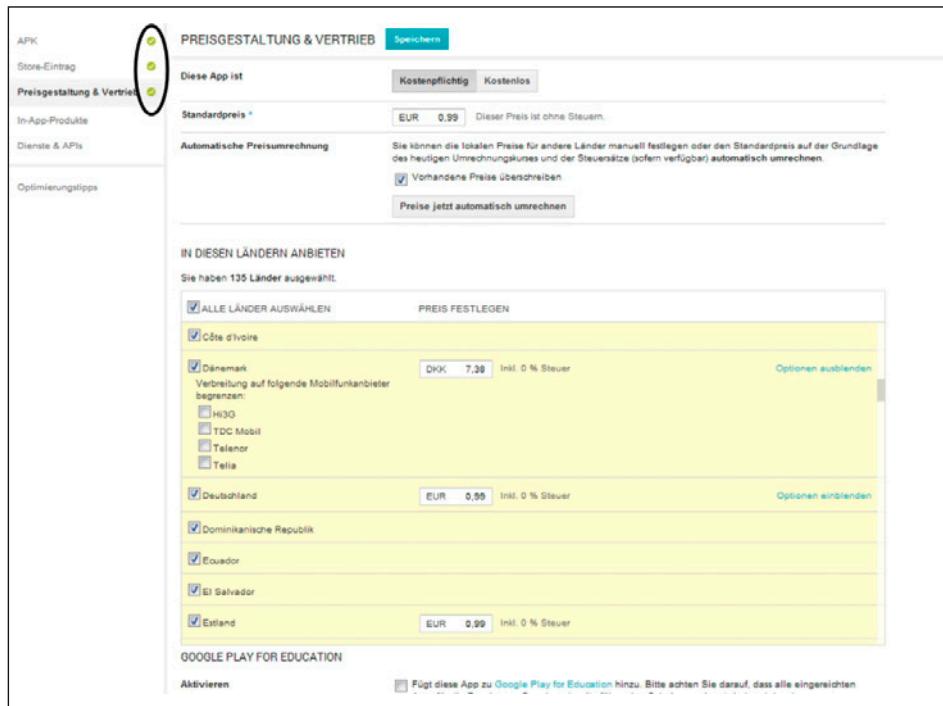


Bild 14.8 Preisgestaltung der App im Play Store

Nachdem nun alle Daten erfasst sind müssen Sie nur noch den Richtlinien und den Exportbestimmungen zustimmen und können die App veröffentlichen. Eventuelle Probleme mit der Veröffentlichung zeigen Ihnen die grünen Icons im Menü-Bereich der Seite an. Haben Sie die App veröffentlicht, dauert es ein paar Stunden, bis Ihre App im Play Store angezeigt wird.

Index

Symbol

/res/drawable 92
- .setBackgroundResource(...) 93

A

AbsolutLayout 69
abstrakte Methoden 60
Activity 30, 70
Activity-Alias 74
Activity for Result 156
- eigene 201
- Result-Code 207
Adapter 99, 101
Address, typisierte Liste 204
ADT-Plug-in
- Einrichtung 14
Aktualisierung 9
AlarmManager 210, 211
AlertDialog 136
- Builder 173
- onDialogPositiveClick() 177
ALPHA-TEST Play Store 227
android\
- background 146
Android Asset Studio 90, 216
Android Dependencies 194
Android DesignGuide 67
Android Lint 220
Android-Manifest, Activity 137
AndroidManifest.xml, Beschreibung 45
android.R, Fehler 194
ANR, application not responsible 169
Api-Key Android-Manifest 198

Api-Key im Manifest 195
ArrayAdapter 106
ArrayList 102, 120
Audio-Dateien, Metadaten 167
AudioManager 168
Audio Stream 168
autoincrement 182

B

Beispielprojekt 4
BETA-TEST Play Store 227
Bezeichner 54
Bitmap, createBitmap 160
BitmapDescriptorFactory 206
- Matrix 160
- Objekt 154
- komprimieren 155
BitmapFactory 159
- decodeFile() 160
- Options 159
Bonuskapitel 9
Broadcast 212
BroadcastReceiver 72, 211
- onReceive() 211
- schließen 128
BufferedWriter 113
Bundle 211
- getExtra() 211

C

Cache-Verzeichnis 166
Calendar 176, 179

Canvas 153
 - onDraw 154
 cast, View 107
 Comparator, vergleichen 117
 CompletionListener 168
 ConnectivityManager 199
 ContentProvider 160
 ContentValues 183
 Context 106
 ConvertView 107
 Cursor Factory, standard 181
 CursorLoader 160
 Cursor-Query 180

D

DatabaseManager 180, 189
 Datei löschen 122
 Dateifilter 117
 Dateiliste sortieren 117
 Dateiname
 - Endung entfernen 127
 - ermitteln 127
 Datenbank
 - Constraints 182
 - DDMS-Ansicht 188
 - Eclipse-Plug-in 188
 - Kommandozeile 187
 - managementsystem 179
 - ROLLBACK 182
 - Spaltenoption 182
 - Update Statement 184
 - whereClause 184
 Datenbankabfragen 180
 Datenmodell 177
 Datensatz, löschen 185
 DatePicker 176
 - Darstellung 176
 - Listener 177
 DateTimeDialogFragment 177
 Datumsformat 120
 - ermitteln 120
 DDMS (Dalvik Debug Monitor) 24
 debug.keystore 196
 Debug-Konfigurationen 51
 Decodierung, Bilder 159
 DefaultCharset 113
 DefaultDisplay 155

density-independent pixel 69
 Dialoge 171
 Dialog-Fragments 171
 DialogInterface 173
 - onClick() 173
 - OnClickListener 173
 Dialog, show() 174
 Display, Orientation 155
 do-while-Schleife 65, 186
 Drawable-Ressource 89, 146

E

Eclipse
 - Einrichtung 14
 Eclipse Plug-in, installieren 188
 EditText 127
 - setText() 127
 Encoder 166
 enum 97, 166
 Enumeration 97
 Equals, Strings vergleichen 139
 Erstellung Datei 112
 Exceptions 66
 ExifInterface 159
 Explizite Intents 83
 Export der App 223
 externer Speicher 112

F

Farbressourcen 146
 FileDate 120
 File, Klasse 112
 FileOutputStream 112
 Format-String 120
 for-Schleife 64
 - mit Iteration 65
 Fragmente 177
 Fragment-Klasse 142
 FragmentManager 204
 Fragments 71
 FrameLayout 69

G

Geocoder 204
 Geoposition LatLng 205

Gesten auswerten 165
 GestureDetector 164
 GestureDetector.OnDoubleTapListener 165
 Google Api Console 195
 - MAP_TYPE 204
 - onClickMap() 205
 - setTitle() 206
 Google Maps Api 191
 Google Maps Api-Key 195
 Google Maps Marker 201
 Google Play Services 191
 Google Play Store 4
 GPS-Sensor 198
 GridLayout 68

H

Hilfsklasse 111

I

IDE = Integrated Development Environment 27
 if, else 63
 ImageView 150
 - erweitert 153
 Implizite Intents 83
 InfoWindowAdapter 204
 InputStreamReader 127
 Intent
 - Activity starten 126
 - Bundle-Daten lesen 127
 - Daten senden 139
 - Email versenden 135
 - expliziter 138
 Intents 82
 - setClass() 126
 - startet Activity 126
 Intent-Filter 211
 - impliziter 135
 - .putExtra() 126
 - putExtra() 210
 - resultIntent 207
 Interface, eigenes 150
 - MultiChoiceModeListener 218
 - OnItemClickListener 108
 interner Speicher 112
 internes Anwendungsverzeichnis 112

J

Java Build Path, Einstellungen 194
 Java-Schlüsselwörter 55
 JDK 11

K

Keystore 196
 keytool 196
 Kodierung, Zeichen 127
 Kontextmenü 85

L

LayoutInflater 177
 LinearLayout 68
 Lint Error Checking 220
 List 107
 ListAdapter 102
 Listen 99
 Listener 179
 ListItem
 - eigenes Layout 105
 - entfernen 219
 - onItemClick() 108
 ListView 101
 - setAdapter() 103
 ListViewCallback 217
 Location-Listener 212
 LocationManager 198, 212
 LocationMapsActivity 198
 Location Provider Criteria 199
 LocationService 198
 LogCat 108
 Logcat Level 62
 LogCat-View 108
 Lokalisierung der App 215
 Long, Datentyp 179

M

MapFragment 202
 Maps Api-Version 2 191
 MediaPlayer 168
 - reset 168
 MediaRecorder 166
 - Ausnahmen 167
 - Konfiguration 166

- Output-Format 166

- Source 166

MediaStore 160

Menü 138

- android

- showAsAction 87

Menü-Item

- aktivieren 122

- deaktivieren 124

Meta Data 74

Metadaten

- Bilder 159

- Media-Datei 167

Minimum Required SDK 29

mkdirs() 112

MotionEvent 165

MultiChoiceModeListener ListView 217

N

Namensraum 68, 147

Name-Space *siehe* Namensraum

Netzwerk-Provider 198

Netzwerkverbindung überprüfen 200

NotificationBar 211

NotificationManager 211

Notifications 211

nullColumnHack 183

O

OnCheckedChangeListener 149

OnItemClickListener 107, 108

onItemClick(), ListItem 126

onMapClickListener 204

onMarkerDragListener 204

onOptionsItemSelected() 87, 142

OnSeekBarChangeListener 150

OnTouchListener 164, 165

Optionsmenü 85

OutputStreamWriter 113

P

Pakete 177

Parcelable 207

Pending-Intent 210

Permissions 46

- SET_ALARM 211

- Google Maps Api 197

persistieren 147

Perspektive

- Eclipse Perspective 32

Play Store Publisher 226

position 204

Positionsdaten in Adressdaten 204

Preisgestaltung 229

Prepared-Statement 180

Preview Play Store 217

PRIMARY KEY 182

Primitive Datentypen in Java\ 54

ProgressBar 167

Projekt

- aus bestehendem Code 192

- Fehler 194

Projekteinstellungen, Bibliotheken 194

Provider 72

ProximityAlert 212

- addProximityAlert() 213

R

Rahmen per xml 91

raw-Dateien, sprachabhängig 220

Receiver 72

Registrierung, Entwickler 195

RelativeLayout 69

Reminder 177

replace() 112

requestCode 157

resultCode 157

Result-Intent 202

Runnable 169

S

sampleSize 160

Schleifen 64, 65

scyte notes 4

SeekBar 146

Selector 220

Serializable 147

Service 73

SHA1-Fingerabdruck 196

Signatur der App 223

SimpleOnGestureListener 165

Spaltendefinition 181
 SparseBooleanArray 219
 SQL-Befehl, CREATE TABLE 181
 SQLite 180
 SQLite, Datentypen 180
 sqlite3 187
 SQLiteOpenHelper 180
 Stream 113
 - Input 127
 - Output 112
 String-Array 173
 String in Integer 159
 String-Ressourcen 88
 Styles 219
 Support Library 202
 SupportFragment 202
 switch case 64
 System-Logger 108

T

Tabellen 180
 - alle Datensätze lesen 186
 TableLayout 68
 Target SDK 29
 Tastaturbefehle, Android-Emulator 49
 tempFile 167
 temporären Datei 166
 Textdatei zeilenweise lesen 128
 Thread 167
 - run 169
 - sleep 169
 Threads 169

TimePicker 176
 - Listener 177
 Toast 212
 Toggle-Buttons 146
 - Status 149
 toString() 117
 Touch Events 163
 touch_move 154

U

UI-Thread 169
 URI 156
 uses-feature 197
 Uses Library 74
 Uses-Permissions Maps 197

V

Verzeichnis rekursiv durchlaufen 120
 Verzeichnisinhalt auflisten 117

W

WebView 139
 while-do-Schleife 65
 Wiedergabe 167

Z

Zugriffsmodifikator 97
 Zugriffsrechte 97

Kompakter Schnelleinstieg für IT-Profis.



Tittel/Baumann

Apps für iOS entwickeln

Am Beispiel einer realen App

234 Seiten.

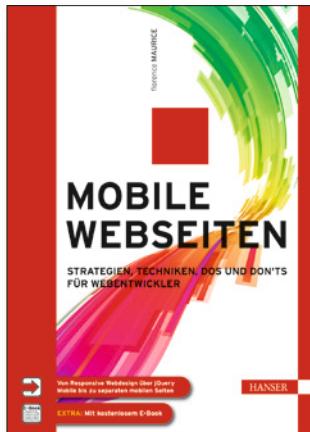
ISBN 978-3-446-43192-8

Dieses Buch stellt Ihnen zunächst die wichtigsten Grundlagen sowie die Syntax der Programmiersprache Objective-C vor. Dann geht's los: In den folgenden 10 Kapiteln entwickeln Sie Schritt für Schritt eine App, mit der Sie Notizen erstellen können in Form von Text, Bild oder Audio. Die jeweiligen Zwischenstände der App (Versionen 0.1 bis 1.0) können Sie im Internet downloaden, so dass Sie an jeder beliebigen Stelle in die Entwicklung einsteigen können. Die fertige App finden Sie unter dem Namen »scyte notes« auch kostenlos im App-Store.

Das App-Beispielprojekt umfasst alle für die professionelle App-Entwicklung wesentlichen Technologien und Features wie z.B.: Arbeiten mit Storyboard, Navigationselementen & Autolayout; Datenanbindung; Ortungsdienste & Karten inkl. Reverse Geocoding; Einsatz von Timer; Veröffentlichung der App.

Mehr Informationen zu diesem Buch und zu unserem Programm
unter www.hanser-fachbuch.de/computer

Groß im Kommen: Mobile Websites!



Maurice

Mobile Webseiten

Strategien, Techniken, Dos und Don'ts

für Webentwickler

416 Seiten. FlexCover.

ISBN 978-3-446-43118-8

→ Auch als E-Book erhältlich

Für die Konzeption einer mobilen Site gibt es eine ganze Reihe von Techniken und grundlegenden Strategien, die Ihnen in diesem praktischen Leitfaden vermittelt werden. Das Buch gliedert sich in drei Themenbereiche:

Teil I Basics beschreibt die Besonderheiten mobiler Geräte, gibt Ihnen einen Überblick über die entsprechenden Browser, erklärt die verschiedenen Strategien für die Erstellung mobiler Sites und beschäftigt sich mit der Inhaltsanordnung.

Teil II Techniken konzentriert sich auf das richtige Markup, auf CSS, Performance-Optimierung, JavaScript und JavaScript-APIs und Strategien für Bilder.

Teil III Umsetzung zeigt Ihnen schließlich, wie Sie per Responsive Webdesign Ihre Website für den mobilen Zugriff optimieren; wie Sie mit den JavaScript-Frameworks jQuery Mobile und Sencha Touch WebApps realisieren; und wie Sie separate mobile Webseiten erstellen.

Mehr Informationen zu diesem Buch und zu unserem Programm
unter www.hanser-fachbuch.de/computer

WPF ganz praxisnah.



Wegener

WPF 4.5 und XAML

Grafische Benutzeroberflächen für Windows
inkl. Entwicklung von Windows Store Apps
704 Seiten.

ISBN 978-3-446-43467-7

→ Auch als E-Book erhältlich

Jörg Wegener beschreibt detailliert die zentralen Elemente der WPF 4.5, ihre zu-grunde liegenden Konzepte und die in WPF implementierte Beschreibungssprache XAML. Zahlreiche Beispiele zeigen Ihnen den professionellen Einsatz des Frame-works in Situationen, mit denen Sie als Entwickler tagtäglich konfrontiert sind.

Einen Schwerpunkt dieser Neuauflage bilden die Neuerungen von WPF 4.5 und Visual Studio 2012. Hier geht es u.a. um die Entwicklung von Apps für Windows 8 mit XAML und der Windows Runtime. Außerdem neu hinzugekommen sind die Themen: Eingabesteuerung via Maus, Tastatur und Touchscreen; das Entwurfs-muster Model-View-View-Model; Installation und Aktualisierung von Anwendungen beim Kunden; Gestaltung mit Expression Blend.

Mehr Informationen zu diesem Buch und zu unserem Programm
unter www.hanser-fachbuch.de/computer

So wird Ihre Website gefunden!



Firnkes

SEO und Social Media

Handbuch für Selbstständige und

Unternehmer

438 Seiten.

ISBN 978-3-446-43550-6

→ Mit kostenlosem E-Book

Ein gelungener Webauftritt alleine genügt nicht für erfolgreiches Online-Marketing. Viel Traffic auf der Website und eine gute Platzierung in Google sind vor allem auch das Ergebnis gezielter SEO- und Social Media-Strategien. Die Suchmaschinenoptimierung wie auch die Kommunikation in sozialen Medien werden jedoch oftmals als Geheimdisziplin verstanden, an die man sich selbst nicht heranwagt, sondern lieber »Experten« beauftragt.

Hier setzt dieser Ratgeber an. Er wendet sich an Selbstständige und Entscheider, die in das Thema einsteigen bzw. ihre Kenntnisse vertiefen wollen. Mit Beispielen aus der Praxis und zahlreichen wertvollen Tipps vermittelt Michael Firnkes die notwendigen inhaltlichen und technischen Grundlagen sowie die Prozesse, um SEO- und Social Media-Strategien selbst umsetzen bzw. souverän lenken zu können.

Mehr Informationen zu diesem Buch und zu unserem Programm
unter www.hanser-fachbuch.de/computer

Fundierter Einstieg in die Spieleentwicklung.



Freiknecht

Spiele entwickeln mit Gamestudio

Virtuelle 3D-Welten mit Gamestudio A8 und Lite-C

480 Seiten. Vierfarbig. FlexCover mit DVD.

ISBN 978-3-446-43119-5

→ Auch als E-Book erhältlich

Dieses umfassende Handbuch zeigt Ihnen, wie Sie die Autorensoftware Gamestudio bedienen, um ein eigenes Spiel zu programmieren. Sie werden sich mit den Editoren vertraut machen, um 3D-Welten zu entwerfen und ihnen mit selbst gemachten Spielfiguren und Gegenständen Leben einzuhauen. Sie werden lernen, wie man ein Projekt so plant, dass es realisierbar ist, gut aussieht und: Spaß macht! Schritt für Schritt wird Ihnen auf spielerische Art und Weise erklärt, wie Sie die Programmiersprache Lite-C und die Funktionen der Gamestudio-API einsetzen. Sie werden 3D-Welten laden, deren Bewohner mit Funktionen versehen und diese auf Klicks mit der Maus, auf ein »Anrempeln«, auf ein Näherkommen des Spielers reagieren lassen. Sie werden eine realistische Naturlandschaft dynamisch mit Pflanzen versehen, zwischen verschiedenen Levels hin und her wechseln, dynamische Ladebildschirme erstellen und und...

Mehr Informationen zu diesem Buch und zu unserem Programm
unter www.hanser-fachbuch.de/computer

PRAXISWISSEN FÜR ENTWICKLER



Testen Sie jetzt

„web & mobile developer“
und sichern Sie sich
2 Ausgaben kostenfrei!

Jetzt anfordern unter:

webundmobile.de/probelesen



APPS FÜR ANDROID ENTWICKELN //

- Für (App-)Entwickler mit OOP-Kenntnissen
- Kompakter Schnelleinstieg in die App-Entwicklung für Android 4.x mit Eclipse
- Vermittelt umfassendes Know-how am Beispiel einer realen App
- Die finale App, die App-Zwischenstände und evtl. Aktualisierungen finden Sie unter:
<http://downloads.hanser.de>

Sie sind Entwickler mit grundlegenden Kenntnissen in moderner objektorientierter Programmierung und/oder App-Entwickler, der bislang aber nicht für Android entwickelt hat? Dieses Buch zeigt Ihnen, wie Sie Ihr Programmierwissen auf Android übertragen, und ermöglicht Ihnen somit einen schnellen und kompakten Einstieg in die App-Entwicklung mit Eclipse und Java.

Es beginnt mit Schritt-für-Schritt-Anleitungen zur Einrichtung der Arbeitsumgebung Eclipse und dem Android SDK sowie mit einem Schnelleinstieg in Eclipse und Java. Learning by doing entwickeln Sie bereits hier Ihre erste kleinere App und testen sie im Emulator sowie auf einem Gerät. Der anschließende Crash-Kurs stellt Ihnen die wichtigsten Grundlagen sowie die Syntax der Programmiersprache Java vor. Dann geht's los: In den folgenden 10 Kapiteln entwickeln Sie eine App, mit der Sie Notizen erstellen können in Form von Text, Bild oder Audio. Die jeweiligen Zwischenstände der App (Versionen 0.1 bis 1.0) können Sie im Internet downloaden, so dass Sie an jeder beliebigen Stelle in die Entwicklung dieser App einsteigen können. Die fertige App finden Sie unter dem Namen »scyte notes« auch kostenlos im App-Store.

Das App-Beispielprojekt umfasst alle für die professionelle App-Entwicklung wesentlichen Technologien und Features wie z.B.: Navigationselemente & Layout; Daten zwischen Activities übergeben; Notifications nutzen; Standortdienste & Karten inkl. Reverse Geocoding; Lokalisierung der App für mehrere Sprachen.

Jan TITTEL ist freiberuflicher Entwickler, Berater und Trainer mit dem Schwerpunkt auf Anwendungen für .NET/Office sowie Windows Phone und iOS.



Jochen BAUMANN verfügt über langjährige Erfahrungen als ERP-/CRM-Berater und ist als Entwickler tätig mit den Schwerpunkten .NET, Java, Datenbanken sowie Android, iOS, Windows Phone.



AUS DEM INHALT //

- Oberflächen & Navigation
- Eigene Klassen, Listen, Adapter
- Mit der ActionBar arbeiten
- Emails & Webseiten
- Toasts einblenden
- Gesten, Grafik & Audios
- Datenbanken erstellen & nutzen
- Google Maps Api V2
- Lokalisierung, Icons, Startbilder
- Veröffentlichung der App

UNSER BUCHTIPP FÜR SIE //



Tittel/Baumann,
Apps für iOS entwickeln
2013, 234 Seiten, FlexCover, € 24,99.
ISBN 978-3-446-43192-8

€ 24,99 [D] | € 25,70 [A]
ISBN 978-3-446-43191-1



9 783446 431911

HANSER