7 Intents

Bereits in Abschnitt 1.4 in Teil I haben wir den Begriff des *Intent* als »Absichtserklärung« kennengelernt. Dort hat eine Activity (`FormularActivity`) eine andere Activity (`ErgebnisActivity`) gestartet. Dies ist der häufigste Einsatzzweck für Intents, nicht jedoch der einzige. Intents sind vielmehr Teil eines Mechanismus zum Austausch von Nachrichten und Daten zwischen Komponenten einer Anwendung, verschiedenen Anwendungen untereinander oder mit der Android-Plattform.

Intents verbinden unabhängige Komponenten, also Activities, Services, Content Provider oder Broadcast Receiver, untereinander zu einem Gesamtsystem und schaffen eine Verbindung zur Android-Plattform. Letzteres erfolgt über sogenannte Broadcast Intents. Es handelt sich dabei um Systemnachrichten, die wir im Kapitel über Systemnachrichten (siehe Kap. 10 auf S. 225) gesondert betrachten.

Wir lernen nun auf einem allgemeinen Niveau das Konzept der Intents kennen. Intents tauchen jedoch im weiteren Projektverlauf immer wieder auf, so dass uns das Thema durch die folgenden Kapitel begleiten wird.

7.1 Warum gibt es Intents?

Komponenten miteinander verbinden

Android ist komponentenbasiert. Da liegt es nahe, die Komponenten mit einem Mechanismus zu verbinden, der standardisiert ist, sich ohne Quellcodeänderung nutzen lässt und einfach zu dokumentieren ist. Auf diese Weise kann eine Komponente von anderen Programmierern genutzt oder einfach gegen eine andere ausgetauscht werden.

Schutz vor unerlaubten Zugriffen

Ähnlich wie die Android-Plattform verwenden die meisten mobilen Plattformen ein Sandbox-Modell. Eine direkte Interaktion zwischen Anwendungen oder Teilen zweier verschiedener Anwendungen ist meist nicht möglich. Bei Android unterliegen die Intents dem Berechtigungssystem. Zugriff auf Komponenten außerhalb der eigenen Anwendung ist daher nicht erlaubt, solange man nicht explizit geeignete Berechtigungen vergibt. Dann jedoch lassen sich auch z. B. Activities oder Services einer anderen Anwendung nutzen.

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

Lose Kopplung von Komponenten

Durch Intents erreichen wir eine lose Kopplung der Bestandteile von Anwendungen. Dies wird durch zwei verschiedene Arten von Intents verschieden stark unterstützt. Wir unterscheiden Intents in explizite und implizite Intents.

7.2 Explizite Intents

Bei einem expliziten Intent ist die Empfängerkomponente bereits bei der Programmierung des Aufrufs bekannt und eindeutig identifiziert.

Klassenname angeben

Die Empfängerkomponente wird im Allgemeinen bei der Erzeugung des Intent durch ihren Namen oder die Klassenangabe übergeben. Wollen wir also eine Activity `PositionSenden` aufrufen, so schreiben wir:

```
final Intent i = new Intent(this, PositionSenden.class);
startActivity(i);
```

Wie kommt man an den Context?

Als ersten Parameter übergeben wir dem Konstruktor des Intents ein Exemplar der Klasse `Context`. Da `android.app.Activity` von `android.content.Context` abgeleitet ist, können wir einfach `this` übergeben. In anderen Fällen, wo wir keinen direkten Zugriff auf den Kontext via `this` haben, hilft die Methode `getApplicationContext` weiter. Wenn wir beispielsweise innerhalb einer Activity ein Exemplar einer anderen Klasse erzeugen und Methoden dieses Exemplars überschreiben, bezieht sich `this` auf das Exemplar und nicht auf die Activiy. Mittels `getApplicationContext` erhält man dann Zugriff auf die Klasse `Context`. Die Methode steht übrigens auch in der Klasse `Service` zur Verfügung, die wir später kennenlernen werden.

Möchte man eine Activity von anderer Stelle aus aufrufen, an der der Kontext nicht zur Verfügung steht, hilft der folgende Code:

```
Intent i = new Intent();
i.setComponent(new ComponentName(
    "de.androidbuch.amando.gui",
    "PositionSenden"));
startActivity(i);
```

Laufzeitfehler möglich

Hier haben wir die Zielklasse über ihren Namen definiert. Der Nachteil dieser Adressierungsform ist, dass die Prüfung auf das korrekte Ziel nicht vom Compiler übernommen werden kann. Fehler treten daher erst zur Laufzeit, das heißt beim ersten Aufruf des Intent, auf.

7.3 Implizite Intents

Implizite Intents adressieren keinen bestimmten Empfänger und überlassen es den Komponenten der installierten Anwendungen, auf den Intent zu reagieren.

Implizite Intents spezifizieren keine Komponente, an die sie adressiert sind. Sie werden praktisch »ins Leere« abgeschickt, in der Hoffnung, dass es eine oder mehrere Komponenten gibt, die mit diesem Intent etwas anfangen können. Meist werden implizite Intents verwendet, wenn man Komponenten anderer Anwendungen nutzen möchte. So ist es beispielsweise möglich, eine Activity aus der *Telefon*-Anwendung (engl. *Dialer*) zum Wählen einer Telefonnummer zu nutzen, ohne sie selbst implementieren zu müssen.

Auf gut Glück

```
Intent intent = new Intent(Intent.ACTION_DIAL,
    Uri.parse("tel:(0228)1234567"));
startActivity(intent);
```

Wie der Programmcode zeigt, wird bei impliziten Intents eine Aktion (hier: `Intent.ACTION_DIAL`) und eine URI (Uniform Resource Identifier) mitgegeben. Während die Aktion der aufzurufenden Komponente mitteilt, was sie tun soll, übermittelt die URI die dafür notwendigen Daten.

Aktion und Datenquelle angeben

Handelt es sich beim Android-Gerät um ein Mobiltelefon, so ist die *»Telefon«*-Anwendung, eine Standardanwendung der Android-Plattform, schon vorinstalliert (engl. *»Dialer«*-Anwendung). Diese *Dialer*-Activity lässt sich über einen impliziten Intent nutzen. Abbildung 7-1 zeigt die Dialer-Activity.

Doch wo ein Sender ist, sollte es auch einen Empfänger geben. Wenn wir Activities anderer Anwendungen aufrufen und in unserem Programm verwenden können, muss es auch die Möglichkeit geben, die Activities unseres Programms anderen Anwendungen zur Verfügung zu stellen. Dazu muss man eine Activity in die Lage versetzen, einen impliziten Intent zu empfangen. Dies geschieht mit Hilfe von *Intent-Filtern*.

Intent-Filter

7.3.1 Intent-Filter für implizite Intents

Intent-Filter werden im Android-Manifest deklariert. Sie stellen quasi den Schlüssel zum Schloss dar. Wenn beispielsweise eine Anwendung A eine impliziten Intent verschickt, kann eine Komponenten in Anwendung B (z. B. eine Activity) mit einen Intent-Filter versehen werden, der auf exakt diesen Intent reagiert. Die Activity aus Anwendung B würde

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

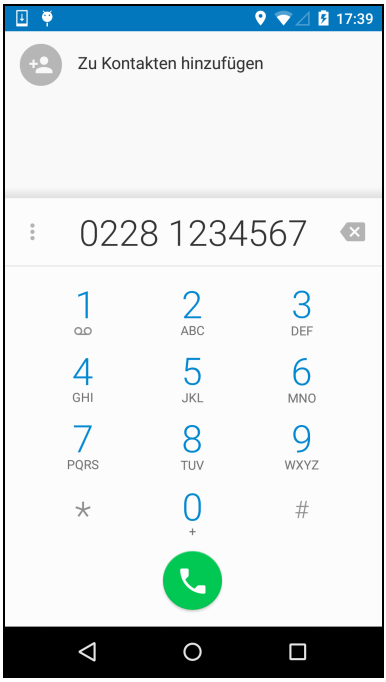

Abb. 7-1
Dialer-Activity

dann so aus Anwendung A heraus gestartet, als wäre die Activity eine Bildschirmseite der Anwendung A.

Einen Intent-Filter für implizite Intents kennen wir schon. Um eine Activity zur Start-Activity unserer Anwendung zu machen, müssen wir einen bestimmten Intent-Filter deklarieren. Beim Start der Amando-Anwendung schickt das Android-Betriebssystem einen impliziten Intent ab. Dieser muss durch die Activit gefangen werden, die die Startseiter der Anwendung darstellt. Beim Anlegen des Amando-Projekts ist automatisch die Activity `Startseite` im Android-Manifest mit dem richtigen Intent-Filter versehen worden.

Listing 7-1
Definition eines Intent-Filters

```
<activity android:name=".gui.Startseite"
          android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name=
        "android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Ein Intent-Filter besteht aus bis zu drei Elementen, mit deren Hilfe er genau festlegen kann, auf welche impliziten Intents er reagiert. Der Aufrufer, der Komponenten per implizitem Intent aufruft, muss man genau

wissen, worauf deren Intent-Filter reagiert und welche Parameter die Komponente erwartet. Daher ist es wichtig, die Intent-Filter gut zu dokumentieren, wenn man Teile seiner Anwendung öffentlich macht.

Intent-Filter gehören dokumentiert.

action Das `<action>`-Element spezifiziert die Aktion, auf die der Intent-Filter reagieren soll. Das `android:name`-Attribut des `<action>`-Elements im Intent-Filters muss eine eindeutige Zeichenkette sein. Sie bezeichnet die Aktion, die stattfinden soll. Selbst definierte Aktionen für Intents sollten der folgenden Konvention entsprechen:

Was soll passieren?

```
<Paketname.der.Anwendung>.intent.action.<ACTION_NAME>
```

Die Klasse `android.content.Intent` besitzt einige generische Action-Intents, nach denen man im Intent-Filter filtern kann. Sie erlauben den Zugriff auf Komponenten, die Bestandteil der Anwendungen der Android-Plattform sind. Beispielsweise konnten wir oben mittels des Action-Intents `Intent.ACTION_DIAL` eine Activity zum Wählen einer Telefonnummer nutzen. Die generischen Action-Intents beginnen mit `ACTION_`.

Androids generische Intents

Eine interessante Sammlung bekannter Intents und deren Adressierung ist unter [4] zu finden. Die Standard-Intents der Android-API sind in der Online-Dokumentation unter [35] aufgelistet.

Übersicht Standard-Intents

category Ein Intent-Filter darf mehrere `<category>`-Elemente besitzen. `<category>` legt fest, wie die Activity aufgerufen wird. Meist wird man hier »`android.intent.category.DEFAULT`« verwenden. Damit teilt man dem Activity Manager mit, dass die Activity normal gestartet werden soll und vollständig auf dem Bildschirm angezeigt werden kann.

Startoptionen für Activities

Im Gegensatz dazu kann man »`android.intent.category.TAB`« verwenden. Damit legt man fest, dass die aufrufende Activity eine `TabActivity` sein muss und die eigene Activity nur als ein Reiter in der Eltern-Activity (also der `TabActivity`) dargestellt wird.

Die Kategorie `android.intent.category.LAUNCHER` gibt es nur für Activities. Sie legt fest, dass die Activity die Startseite einer Anwendung ist. Diese Kategorie kennen wir schon aus dem Android-Manifest.

Startseite festlegen

Als letztes Beispiel sei noch »`android.intent.category.PREFERENCE`« erwähnt. Damit kann man eine `PreferenceActivity` nach außen zur Verfügung stellen, damit andere Anwendungen sie in ihre Activities zur Pflege der Programmeinstellungen einbinden können.

Es gibt noch einige weitere Kategorien, die der API-Dokumentation zu entnehmen sind.

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

Mit welchen Daten?

data Beim Abschicken eines Intents kann ihm eine URI mitgegeben werden. Diese URI spezifiziert eine Datenquelle. Wir haben oben schon einen Intent mit einer URI kennengelernt: `new Intent(Intent.ACTION_DIAL, Uri.parse("tel:(0228)1234567"))`. Im Intent-Filter verwendet man das `<data>`-Element, damit nur Intents mit ganz speziellen URIs eine bestimmte Komponente aufrufen. Daher bieten die Attribute des `<data>`-Elements alle Möglichkeiten, eine URI zu analysieren. Nun stellen wir die entsprechenden Attribute vor, die es ermöglichen, nach den Bestandteilen einer URI zu filtern:

- `android:scheme` Gibt das Schema für den Zugriff auf die Daten an. Im Zusammenhang mit Android sind `content`, `http`, `geo` und `file` häufig gebrauchte Werte. Mindestens das `scheme`-Attribut muss innerhalb von `data` definiert sein. Es können ein oder mehrere `scheme`-Attribute gesetzt werden.
- `android:mimetype` Angabe eines Mime-Typs. Man kann selbst definierte oder bekannte Mime-Typen verwendet werden. Zulässig ist z. B. `image/jpeg` oder der selbst definierte Mime-Typ `vnd.androidbuch.cursor.item/*`. Das Sternchen (»Asterisk«) bedeutet, dass jeder Subtyp akzeptiert wird.
- `android:host` Angabe eines Hostnamens (z. B. »`developer.android.com`«)
- `android:path` Setzt voraus, dass das `host`-Attribut gesetzt wurde. Gibt den Pfad zu den Daten an. Im Falle einer URL z. B. `reference/android/content/Intent.html`. Will man mehr Flexibilität und Zugriff auf bestimmte Daten in Teilbäumen der Pfadstruktur zulassen oder mit Wildcards arbeiten, kann man statt `path` die Attribute `pathPrefix` oder `pathPattern` verwenden. Näheres zu deren Verwendung findet sich in der Dokumentation des Android-SDK ([21]).
- `android:port` Angabe eine Ports (z. B. 8080). Wird ignoriert, wenn `host` nicht gesetzt wurde.

`category` und `data` sind optional. Mittels `category` wird in der Regel unterschieden, welcher Art der Aufruf sein soll. Typische Werte entsprechen umgangssprachlich »Standard«, »Anzeigen« oder »Editieren«. Einer Activity kann so beispielsweise mitgeteilt werden, dass sie einen Text entweder im Bearbeitungsmodus oder im Anzeigemodus darstellen soll. Das `data`-Attribut teilt der Komponente mit, von welchem Typ die Daten sind. Dies entspricht in etwa dem MIME-Type bei einem Zugriff via Webserver.

7.3.2 Ein Beispiel

Um den Umgang mit Intent-Filtern zu verdeutlichen, definieren wir einen Intent-Filter und den dazu passenden impliziten Intent. Den Intent-Filter fügen wir der Deklaration einer Activity im Android-Manifest hinzu. Die Werte für die XML-Attribute `action`, `category` und `data` wählen wir für unsere Zweckes passend. Das `data`-Attribut ist optional.

```
<activity android:name=".gui.DemoActivity">
  <intent-filter>
    <action android:name=
      "de.visionera.androidbuch.demo.intent.
        action.DEMO_INTENT" />
    <category android:name="
      android.intent.category.DEFAULT" />
    <data android:scheme="http"
      android:host="www.visionera.de" >
  </intent-filter>
</activity>
```

Um die Bildschirmseite `DemoActivity` per implizitem Intent aufzurufen, müssen die Action-, Category- und Data-Parameter passend gewählt werden. Den Action-Parameter haben wir entsprechend der oben erwähnten Konvention erstellt. Es handelt sich lediglich um einen String. Gleiches gilt für die Category, jedoch haben wir hier einen Wert verwendet, den uns das Android-System bereitstellt. Die Klasse `Intent` kennt einige Konstanten vom Typ String, die als Category-Attribut verwendet werden können. Man kann sich jedoch auch eigene Werte ausdenken. In unserem Fall handelt es sich um den Wert `android.intent.category.DEFAULT`, den die Klasse Intent durch die Konstante `Intent.CATEGORY_DEFAULT` bereitstellt.

```
Intent intent = new
  Intent("de.visionera.androidbuch.demo.intent.
      action.DEMO_INTENT",
      Uri.parse("http://www.visionera.de/"));
intent.addCategory(Intent.CATEGORY_DEFAULT);
startActivity(intent);
```

Der implizite Intent erhält als ersten Parameter im Konstruktor den String für die Action. Den gleichen String haben wir im Intent-Filter verwendet. Als `data`-Attribut verwendet wir eine Internet-Adresse. Der Intent-Filter reagiert nur, wenn das Schema »http« lautet und der Host »www.visionera« entspricht. Für die Kategorie verwenden wir die oben erwähnte Konstante der Klasse Intent.

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

Damit kennen wir die Grundlagen von expliziten und impliziten Intents. Nun werden wir noch erklären, wie der Filterprozess bei impliziten Intents abläuft.

7.3.3 Intent-Resolution

Empfänger ermitteln

Als *Intent-Resolution* wird der Prozess bezeichnet, den Android durchführt, um implizite Intents aufzulösen. Ziel dabei ist es, die Komponente zu finden, die am besten auf den Intent passt. Diese wird dann gestartet. In der Praxis wird man sich häufiger mal wundern, warum die gewünschte Komponente nicht startet. Dann hilft es, die Regeln der *Intent-Resolution* zu kennen, um den Fehler zu finden.

Regeln der Intent-Resolution

- **Regel 1**: Der Intent passt nicht zur Komponente, wenn keines der `action`-Elemente des Intent-Filters zum Intent passt.
- **Regel 2**: Der Intent passt nicht zur Komponente, wenn er Kategorien enthält, für die es kein `category`-Element im Intent-Filter gibt. Für alle Kategorien des Intent muss es Entsprechungen im Filter geben.
- **Regel 3**: Die URI des Intent muss mindestens mit einem der `data`-Elemente übereinstimmen. Wurde im Intent keine URI als Verweis auf die Datenquelle angegeben, passiert der Intent den Intent-Filter und die Komponente wird aufgerufen.
- **Regel 4**: Werden zu einem Intent mehrere Komponenten gefunden, waren deren Intent-Filter vielleicht zu allgemein gefasst. Android präsentiert dann einen Bildschirm, der eine Auswahl der gewünschten Komponente zulässt (siehe Abb. 7-2).

7.3.4 Beispiele für implizite Intents

Schauen wir uns einige gängige Beispiele an, wie man Komponenten der Android-Standardanwendungen nutzt. Zu den Standardanwendungen gehören unter anderem:

- Telefon: Zum Aufbau von Telefonverbindungen
- Maps: Anzeige von Karten mit `MapsActivity`
- E-Mail: Standard-Mail-Client
- Kontakte: Eigene Kontakte verwalten
- Browser: Standardbrowser auf Basis von WebKit

Abb. 7-2
Android-Dialog zur Auswahl der Komponente

- Market: Android-Market-Anwendung zur Installation von weiteren Anwendungen von Drittanbietern
- Kamera: Aufnehmen von Bildern und Videos
- ...

Eine Nummer wählen

Nummer direkt wählen

Im vorherigen Abschnitt haben wir schon gesehen, wie man aus der eigenen Anwendung heraus eine Activity zum Starten eines Telefonanrufs nutzen kann. Man kann einen Anruf auch absetzen, so dass die Nummer sofort und ohne Rückfrage gewählt wird. Dazu muss nur `Intent.ACTION_CALL` statt `Intent.ACTION_DIAL` angegeben werden.

```
Intent intent = new Intent(Intent.ACTION_CALL,
    Uri.parse("tel:(+49) 228 1234567"));
startActivity(intent);
```

Der implizite Intent startet eine Activity der Telefon-Anwendung (siehe Abb. 7-3), die den Versuch eines Verbindungsaufbaus mit der gewählten Telefonnummer anzeigt.

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

Abb. 7-3
Eine Telefonnummer anrufen

Position in einer Karte anzeigen

Zwei Arten des Aufrufs

Die Standardanwendung *Maps* kann zur Darstellung einer Position auf einer Landkarte genutzt werden. Dabei haben wir zwei unterschiedliche Arten des Aufrufs zur Verfügung. Zum einen können wir den Längen- und den Breitengrad angeben, um die Karte rund um diese Position anzuzeigen. Zum anderen können wir einen Suchstring mit einer Adressangabe mitgeben. Schauen wir uns den ersten Anwendungsfall an:

```
Uri uri = Uri.parse("geo:50.7066272,7.1152637?z=19");
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(uri);
startActivity(intent);
```

Positionsdaten übergeben

Dieser Intent startet eine Activity der Anwendung *Maps*, wie in Abbildung 7-4 zu sehen. Wir haben die URI für den Ortspunkt über die Methode `setData` gesetzt und nicht über den Konstruktor des Intent. Dadurch wird deutlicher, dass wir mit der URI Daten an die Activity übergeben. Man beachte auch den zusätzlichen Parameter `z=19`. Hiermit stellt man den Zoomlevel der Kartenanzeige ein. Der Maximalwert beträgt 23. Ein minimaler Zoomlevel zeigt den ganzen Erdball.

Adressen als Parameter übergeben

Ganz ähnlich wie in Abbildung 7-4 sieht das Ergebnis aus, wenn man eine Adresse als Parameter einer Suche übergibt. Findet Google Maps die Adresse, so kennt es die zugehörige Position und sendet

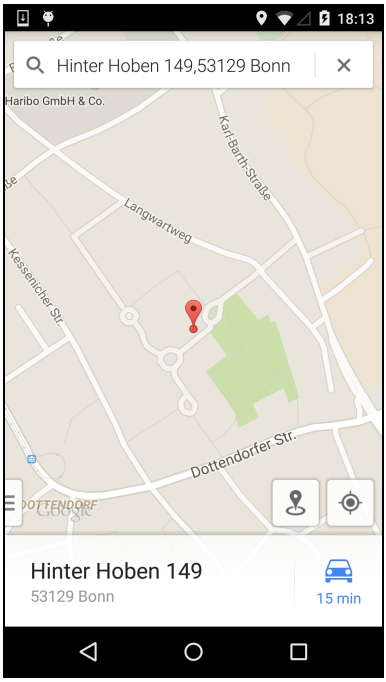

Abb. 7-4
Google-Maps-Activity

das passende Kartenmaterial an die Maps-Activity. Mit dem folgenden Quellcode übergeben wir eine Adresse:

```
Uri uri = Uri.parse("geo:0,0?" +
    "q=Hinter%20Hoben%20149,53129%20Bonn");
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(uri);
startActivity(intent);
```

Längen- und Breitengrad müssen dabei jeweils Null sein. Leerzeichen in der Adresse werden durch »%20« ersetzt.

Google Street View

3-D-Ansicht

Wird die Maps-Anwendung mit dem Schema `geo` aufgerufen, erhalten wir eine Kartendarstellung aus der Vogelperspektive. Mit der Einführung von *Google Street View* steht für manche Gegenden mancher Länder eine 3-D-Ansicht der Erdoberfläche zur Verfügung, die bis auf Straßenebene herunterreicht. Da Google die Straßen mit speziellen Fahrzeugen mit 3-D-Kameras abfährt, ist der Detailreichtum des Bildmaterials für Street View enorm.

Update nötig?

Die Maps-Anwendung ist erst seit der Version 3.1.0 in der Lage, Street-View-Bildmaterial anzuzeigen. Wenn wir das aktuelle Android-SDK installiert haben, können wir Street View nutzen. Installiert ein

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

Anwender das Programm auf einem Android-Gerät, muss dieser dafür sorgen, dass die Maps-Anwendung mindestens die Version 3.1.0 hat. Ist dies nicht der Fall, muss er ein Update über den Google Play Store durchführen.

```
Uri uri = Uri.parse(
    "google.streetview:cbll=44.640381,-63.575911");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

Abb. 7-5
Google Street View verwenden

Mittels cbll die Position übergeben

Als Schema wird in der URI `google.streetview` verwendet. Mit dem Parameter `cbll` übergibt man wieder den Breiten- und den Längengrad. Es gibt einige weitere optionale Parameter, die Blickrichtung und den Zoomlevel betreffend. Wir empfehlen einen Blick in die Google-Dokumentation [20].

Eine Webseite anzeigen

Mit dem folgenden Intent starten wir eine Activity der Android-Standardanwendung »*Browser*« und rufen darin die Internetadresse `www.visionera.de` auf. Abbildung 7-6 zeigt das Ergebnis des Intents.

```
Uri uri = Uri.parse("http://www.visionera.de");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

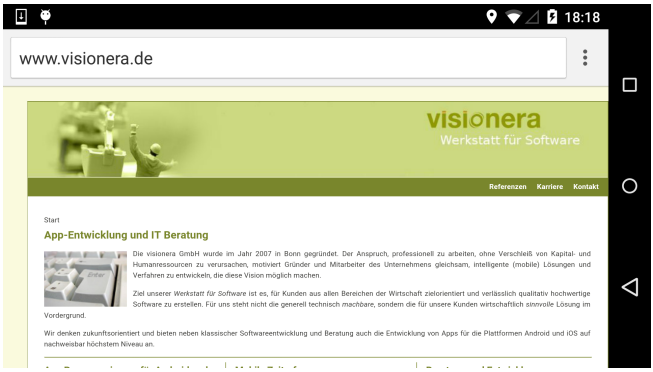

Abb. 7-6
Eine Webseite im Browser anzeigen

Navigation mittels implizitem Intent

Die Buchanwendung Amando bietet in der Activity `KarteAnzeigen` unter dem Options-Menüpunkt »Navigation starten« ein Beispiel für die folgende Implementierung.

Ist das GPS-Modul eingeschaltet, kann man sich per Google-Navigation (siehe Abb. 7-7) von der eigenen Position zur einer entfernten Position navigieren lassen. Dazu bedarf es nur der Angabe des Zielorts. Diese Angabe kann als Adresse oder direkt als Geokoordinate erfolgen, wie es das folgende Listing zeigt:

```
final String geoKontaktPosition = "50.634563,7.112345";

final Intent naviIntent = new Intent(Intent.ACTION_VIEW,
    Uri.parse("google.navigation:q=" +
        geoKontaktPosition));
startActivity(naviIntent);
```

Navigationshilfe mit Google Maps

Manchmal reicht auch eine Routenbeschreibung per Goole Maps aus. Der Vorteil ist, dass hierfür kein GPS-Signal benötigt wird, was für Fussgänger in Häuserschluchten schon mal von Vorteil sein kann. Google Maps kann zur Anzeige von Routeninformationen mit Hilfe einer URI der Form `http://maps.google.com/maps?saddr=<breitengrad>,<längengrad>&daddr=<breitengrad>,<längengrad>` aufgerufen werden, dabei werden im Parameter `saddr` die Start-, und im Parameter `daddr` die Zielkoordinaten mitgegeben. Der folgende Code zeigt den Aufruf per Intent:

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

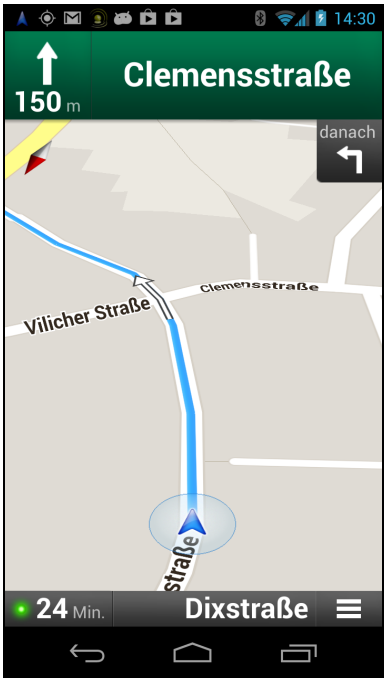

Abb. 7-7
Anzeige der Route mit Google Navigation

```
final String geoKontaktPosition = "50.634563,7.112345";
final String meinePosition = "50.880000,7.060000";

final Intent gmapsIntent = new Intent(Intent.ACTION_VIEW,
    Uri.parse("http://maps.google.com/maps?saddr=" +
meinePosition +
"&daddr=" +
geoKontaktPosition));
startActivity(gmapsIntent);
```

Dies sieht nach einer gewöhnlichen Internetadresse aus. Die Android API ab Version 2.1 erkennt die Adresse jedoch beim Absetzen des Intents und bietet als Alternative den Aufruf der Google-Maps-Anwendung an. Das Ergebnis sehen Sie in Abbildung 7-8.

Ältere Google-Maps-Anwendungen können die berechnete Route nicht in der Karte anzeigen. Sie können jedoch die Route berechnen und als Text ausgeben. Um die berechnete Route auf einer Karte zu sehen, wird der Browser aufgerufen.

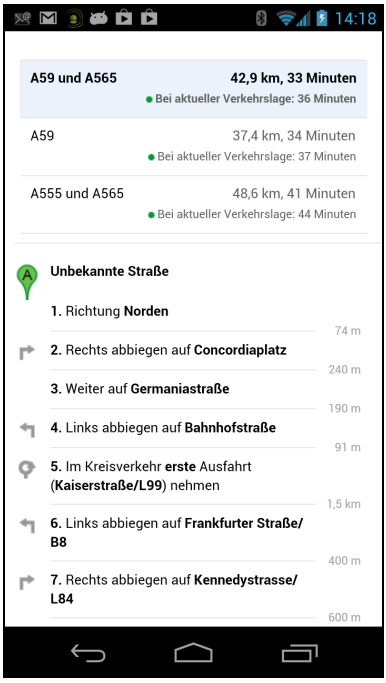

Abb. 7-8
Anzeige der Route mit Google Maps

7.3.5 Fehlerbehandlung

Es kann vorkommen, dass die Komponente, die unser Intent aufrufen soll, gar nicht existiert, z. B. weil wir die dazugehörige Anwendung gar nicht auf unserem Android-Gerät installiert haben. Falls keine passende Zielkomponente gefunden wird, wird ein Laufzeitfehler (`ActivityNotFoundException`) ausgelöst.

Hört mich jemand?

Um dem vorzubeugen, ist es gut, wenn man bereits vor dem Aufruf des Intent weiß, ob die Zielkomponente auf dem Gerät installiert ist. Mit dieser Kenntnis könnte man die `ActivityNotFoundException` vermeiden. Denn diese Exception wird nicht innerhalb der eigenen Anwendung, sondern innerhalb des Package Managers geworfen, der Teil der Android-Plattform ist. Der Intent wird zwar fehlerfrei aus der Anwendung heraus verschickt, die Android-Plattform kann jedoch keinen Empfänger finden und wirft die `ActivityNotFoundException`. Das Ergebnis ist eine Meldung auf dem Bildschirm des Android-Geräts, wie sie in Abbildung 7-9 zu sehen ist.

Vor dem Verschicken prüfen

Listing 7-2 demonstriert, wie man innerhalb einer Activity prüfen kann, ob es einen Empfänger für den Intent gibt. Dazu befragt man den Package Manager, den man über den `Context` der Anwendung (`this`) erhält. Er verwaltet die Intent-Filter aller installierten Anwendungen und kann somit prüfen, ob der Intent ins Leere geht.

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

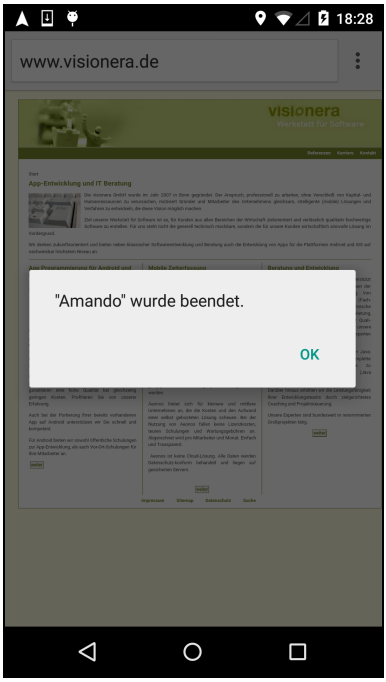

Abb. 7-9
Ergebnis eines Intent ohne Empfänger

Listing 7-2
Verfügbarkeitsprüfung für Intents

```
private boolean isIntentErreichbar(Intent intent) {
  final PackageManager pm = this.getPackageManager();
  List<ResolveInfo> list =
      pm.queryIntentActivities(intent,
          PackageManager.MATCH_DEFAULT_ONLY);
  return list.size() > 0;
}
```

Fassen wir zusammen: Wir haben bisher Intents und Intent-Filter kennengelernt. Wir wissen, dass wir vorhandene Komponenten verwenden können. Dabei ist es egal, ob wir sie selbst implementiert haben oder ob sie zu einer anderen Anwendung gehören, die auf dem Android-Gerät installiert ist. Wenn wir den Intent-Filter der Komponente kennen, die wir verwenden wollen, können wir einen Intent implementieren, der vom Intent-Filter dieser Komponente durchgelassen wird.

7.4 Sub-Activities

Eine Sub-Activity ist zunächst einmal eine ganz normale Activity. Sie wird im Android-Manifest genauso wie jede andere Activity deklariert. Sub-Activities werden aber aus anderen Activities heraus gestartet und bleiben mit diesen während ihrer Lebensdauer verbunden. Über einen

Callback-Mechanismus kann die Sub-Activity Daten an die aufrufende Activity zurückgeben, bevor sie beendet wird.

7.4.1 Sub-Activities aufrufen

Details anzeigen

Der häufigste Einsatzzweck für Sub-Activities sind Master-Detail-Szenarien. Beispielsweise stellt die Activity alle Kontakteinträge als Auswahlliste dar. Wählt der Anwender einen bestimmten Kontakt aus, startet die Anwendung eine Sub-Activity, in der man den Kontakt editieren kann. Speichert man die Änderungen an dem Kontakt, schließt sich die Sub-Activity, und die Activity mit der Kontaktliste muss aktualisiert werden. Eine Sub-Activity startet man mittels der Methode `startActivityForResult`, die einen Intent absetzt.

```
Intent intent = new Intent(Intent.ACTION_PICK,
    ContactsContract.Contacts.CONTENT_URI);
startActivityForResult(intent, 101);
```

Achtung!

Mit Android 2.0 hat sich die »Kontakte«-Anwendung grundlegend geändert. Die zugrunde liegende Datenstruktur ist komplett ausgetauscht worden. Einige Klassen wurden als »deprecated« markiert. Dies hat zur Folge, dass die Implementierung des obigen Intent nur ab Android 2.0 kompilierbar und lauffähig ist.

Da es noch einige wenige Geräte gibt, die unter Android 1.5 oder 1.6 laufen, geben wir hier zusätzlich den Code für Android 1.x an:

```
Intent intent = new Intent(Intent.ACTION_PICK,
    People.CONTENT_URI);
startActivityForResult(intent, 101);
```

Kontakte bearbeiten

Im obigen Fall wird eine Activity aus der vorinstallierten Android-Anwendung *Contacts* zum Bearbeiten eines Kontakts gestartet.. Damit der Aufruf ohne Exception funktioniert, muss dem Android-Manifest wie in Abschnitt 3.3 beschrieben die folgende Berechtigung hinzugefügt werden:

```
<uses-permission android:name=
    "android.permission.READ_CONTACTS"/>
```

In der Methode `startActivityForResult` übergibt man als zweiten Parameter einen Request-Code. Der Request-Code wird später zusammen mit einem Result-Code per Callback an die aufrufende Activity zurückgegeben. Beim Aufruf von Sub-Activities mittels `startActivityForResult` muss der Request-Code >= 0 sein. Andernfalls wird die Activity wie eine normale Activity ohne Callback-Mechanismus aufgerufen.

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

7.4.2 Sub-Activities verwenden

Wir können auf die gleiche Weise Activities unseres eigenen Programms als Sub-Activities nutzen. Wir zeigen den gesamten Zyklus vom Aufruf bis zum Callback samt Parameterübergabe an einem Beispiel aus der Amando-Anwendung.

Programmablauf Beim Druck auf die Schaltfläche »Position senden« auf der Startseite soll die neue Activity `PositionSenden` aufgerufen werden. Sie soll jedoch zunächst als Sub-Activity die schon in Kapitel 6.3 implementierte Activity `GeoKontakteAuflisten` aufrufen. Dort wird aus der Liste der Geokontakte einer ausgewählt und seine Id an die aufrufende Activity (`PositionSenden`) zurückgegeben. Dort sollen zur Kontrolle einige Daten des Geokontakts angezeigt werden (Name, Telefonnummer, letzte bekannte Position). Mittels einer Checkbox kann man festlegen, ob man die eigene Position nur einmalig an den Geokontakt übertragen möchte und ob sie ab dem Absendezeitpunkt fortwährend für den Geokontakt aktualisiert wird. Eine Schaltfläche ermöglicht das Versenden der eigenen Position an den Geokontakt (siehe Abb. 7-10).

Sub-Activity liefert Id

Um diesen Workflow zu implementieren, müssen wir mehrere Punkte abarbeiten:

- Activity `de.androidbuch.amando.gui.PositionSenden` anlegen
- `PositionSenden` ins Android-Manifest eintragen
- ein Layout für `PositionSenden` definieren
- der Activity `PositionSenden` das Layout zuweisen
- Aufruf von `PositionSenden` ermöglichen
- Aufruf der Sub-Activity `GeoKontakteAuflisten` implementieren
- Callback in `GeoKontakteAuflisten` implementieren
- Callback in `PositionSenden` auswerten

Zunächst erzeugen wir im Amando-Projekt eine neue Klasse `de.visionera.androidbuch.amando5.gui.PositionSenden` und tragen sie ins Android-Manifest ein. Das Layout für die Activity `PositionSenden` haben wir im Anhang 25.2.2 auf Seite 521 abgedruckt. Wir fügen es unter dem Namen `position_senden.xml` den Layout-Ressourcen hinzu und ergänzen die Referenzen auf die Texte in der Datei `strings.xml`. Auch diese Datei finden Sie im Anhang auf Seite 519. Anschließend implementieren wir die `onCreate`-Methode der `PositionSenden`-Activity und setzen dort das Layout (siehe Listing 7-3).

Listings im Anhang

Dann sorgen wir dafür, dass die Activity aufgerufen wird, sobald wir auf der Startseite auf die Schaltfläche »Position senden« drücken. Das Layout der Startseite (siehe Listing 5-32 auf S. 105) gibt uns den

Hinweis, dass wir noch die Methode `onClickPositionSenden` im Quellcode der Startseite implementieren müssen:

```
public void onClickPositionSenden(final View sfNormal) {
  final Intent i = new Intent(this, PositionSenden.class);
  startActivity(i);
}
```

Abbildung 7-10 zeigt die Activity in Aktion. Wir können uns nach diesen Vorarbeiten dem Aufruf der Sub-Activity widmen.

Abb. 7-10
Die Activity PositionSenden in Aktion

Gleich beim Erzeugen der Activity soll die Sub-Activity aufgerufen werden. Daher fügen wir den dafür nötigen Programmcode in die `onCreate`-Methode der `PositionSenden`-Activity ein (Listing 7-3). Hier haben wir noch eine weitere Besonderheit, die Intents betrifft. Die Activity `GeoKontakteAuflisten` hat zwei Funktionen. Zum einen dient sie der Auswahl eines Geokontakts, wenn man im Hauptmenü mittels eines Klicks auf die Schaltfläche »Position senden« die eigene Position an einen Freund senden möchte. Zum anderen gibt es im Hauptmenü der Anwendung den Menüpunkt »Geokontakte verwalten«. Auch hier wird die Liste der Geokontakte aufgerufen, aber mit dem Zweck, einzelne Kontakte zu bearbeiten oder Kontakte aus dem Adressbuch des Android-Geräts zu importieren.

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

Damit wir beide Fälle unterscheiden können, wollen wir beim Aufruf der GeoKontakteAuflisten-Activity einen Parameter mitgeben, der diese Unterscheidung möglich macht. Dazu bietet die Klasse Intent die Methode putExtra. Mit ihrer Hilfe kann man dem Intent Schlüssel-Wert-Paare hinzufügen, die in der Ziel-Activity wieder ausgelesen werden können.

Listing 7-3
GeoKontakteAuflisten als Sub-Activity aufrufen

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  setContentView(R.layout.position_senden);

  final Intent intent = new Intent(this,
      GeoKontakteAuflisten.class);
  intent.putExtra(GeoKontakteAuflisten.SELECT_KONTAKT,
      true);

  startActivityForResult(intent, 0);
}
```

Nun passen wir die Activity »GeoKontakteAuflisten« an, so dass sie die Geokontakt-Id an PositionSenden zurückgibt. Dabei ergänzen wir auch die verwendeten String-Konstanten, die die Schlüsselnamen enthalten. Aus Listing 6-4 auf Seite 133 wissen wir, wie man auf eine Listenauswahl (Methode onListItemClick) reagiert.

Listing 7-4
Ein Geokontakt wurde ausgewählt…

```
public class GeoKontakteAuflisten extends ListActivity {
  static final String SELECT_KONTAKT = "SELECT_KONTAKT";
  static final String IN_PARAM_KONTAKT_ID = "KONTAKT_ID";
  ...
  @Override
  protected void onListItemClick(ListView l, View v,
      int position, long id) {
    super.onListItemClick(l, v, position, id);

    final Intent intent = new Intent();
    intent.putExtra(IN_PARAM_KONTAKT_ID, id);
    setResult(Activity.RESULT_OK, intent);
    finish();
  }
  ...
}
```

Id im Intent zurückgeben

Der Callback zurück zur aufrufenden Activity enthält einen Intent, der in diesem Fall als Container für die Rückgabewerte dient. Hier lohnt sich ein Blick in die API-Dokumentation, denn es gibt zahlreiche put-

Methoden für die Rückgabewerte. In der Methode `setResult` setzt man noch einen Statuswert, hier `Activity.RESULT_OK`. Im Fehlerfall gibt man `RESULT_CANCELED` zurück. Der Aufruf von `finish` beendet die Sub-Activity `GeoKontakteAuflisten`. Dies ist nicht nötig, gibt aber Ressourcen frei, da wir die Activity vermutlich so schnell nicht mehr brauchen.

Die Activity `PositionSenden` erweitern wir nun noch um eine Methode, um den Callback empfangen zu können.

Listing 7-5
Ergebnis der Sub-Activity auswerten

```
public class PositionSenden extends Activity {
  private long mGeoKontaktId;
  ...
  @Override
  protected void onActivityResult(int requestCode,
      int resultCode, Intent data) {
    switch (resultCode) {
      case Activity.RESULT_OK:
        mGeoKontaktId = data.getExtras().getLong(
            GeoKontakteAuflisten.IN_PARAM_KONTAKT_ID);
        break;
      case Activity.RESULT_CANCELED:
        finish();
        break;
      default:
        // Fehlerbehandlung...
        break;
    }
    super.onActivityResult(requestCode, resultCode, data);
  }
}
```

In Listing 7-5 wird der Result-Code ausgewertet. Ist alles in Ordnung, wird die Id des Geokontakts aus dem Intent extrahiert und als Attribut der Klasse gespeichert. Doch wie kommt der Kontakt nun zur Anzeige? Hier greifen wir etwas vor. Beim Start einer Activity werden mehrere Methoden durchlaufen. Eine dieser Methoden haben wir schon kennengelernt, die `onCreate`-Methode. Eine weitere Methode ist die `onStart`-Methode. Sie wird immer aufgerufen, wenn eine Activity auf dem Bildschirm angezeigt wird. In Kapitel 15 lernen wir mehr über die Bedeutung dieser Methoden.

Kontakt anzeigen

Für uns ist hier nur entscheidend, dass die Methode *nach* der Methode `onActivityResult` aufgerufen wird. Damit ist die `onStart`-Methode der richtige Platz, um anhand von `mGeoKontaktId` den Geokontakt aus der Datenbank zu laden und ihn anzuzeigen.

Reihenfolge der Aufrufe

Arno Becker / Marcus Pant, Android 5, dpunkt.verlag, ISBN 978-3-86490-260-4

```
@Override
protected void onStart() {
  // TODO: Lade Geokontakt aus der Datenbank
  //       und zeige ihn an.

  super.onStart();
}
```

Zurück zur Praxis

Wir werden in den folgenden Kapiteln noch häufiger in Theorie und Praxis mit Intents konfrontiert werden.

7.5 Fazit

Wir haben in diesem Kapitel den »Leim« kennengelernt, der die einzelnen Komponenten einer Android-Anwendung zu einem Ganzen verbindet: den Intent. Mit Hilfe dieses Konstrukts können die Komponenten sich gegenseitig starten und Daten austauschen. Dabei können durchaus auch Anwendungsgrenzen überschritten werden, und wir können Komponenten anderer Anwendungen in unserer Anwendung verwenden. Wir haben zahlreiche Beispiele für implizite Intents kennengelernt, die uns die Möglichkeit geben, unsere eigene Anwendung mit wenigen Programmzeilen um umfangreiche Funktionen zu erweitern.