6 Oberflächen und Daten

Nachdem wir nun viel über die Gestaltung von Oberflächen und den Zugriff auf Views gelernt haben, möchten wir in diesem Kapitel zeigen, wie man Bildschirmseiten mit Daten aus einer Datenquelle füllt. Die Datenquelle liefert uns beispielsweise Massendaten, die wir in einer Liste darstellen wollen. Oder wir haben Zugriff auf eine Bildgalerie und möchten alle Bilder darstellen. Dazu müssen wir eine Verbindung zwischen den Daten und der Oberfläche herstellen, ohne jeden einzelnen Datensatz selbst verarbeiten zu müssen.

6.1 Zielsetzung

In Amando können wir Bekannten unsere Position mitteilen. Dazu müssen wir aus einer Liste aller der Anwendung bekannten Personen eine bestimmte Person (Geokontakt genannt) auswählen. Anhand dieses und weiterer Fälle werden wir zeigen, wie man eine Verbindung zwischen Datenquelle und Oberfläche mit Hilfe von *Adaptern* herstellt. Wir werden lernen, wie man diese mit den dafür passenden View-Elementen, den *AdapterViews*, verknüpft. Nicht immer ist jedoch ein Adapter nötig. Für Drop-down-Boxen mit immer gleichen Texten können auch Array-Ressourcen als Datenquelle für die Auswahltexte verwendet werden.

Adapter verbinden Daten und Views.

Um auch große Datenmengen performant und ressourcenschonend darstellen zu können, werden wir zeigen, wie man eigene Adapter implementiert. Mittels Callback-Methoden kann auf die Listenauswahl des Anwenders reagiert werden. Im Anschluss zeigen wir, wie man Activities für die Grundeinstellungen einer Anwendung programmiert und die eingegebenen Daten speichert. Speichervorgänge können länger dauern, weshalb wir abschließend die Verwendung von Fortschrittsanzeigen behandeln.

Verarbeitung von Massendaten

6.2 AdapterViews und Ressourcen

Layout anlegen

Wir legen zuerst das Layout für die neue Activity `GeoKontakteAuflisten` fest. Sie wird auf der Startseite durch einen Klick auf die Schaltfläche »*Geokontakte*« aufgerufen. Wir legen die Activity im Package `gui` an und tragen sie ins Android-Manifest ein. Listing 6-1 zeigt das Layout `geokontakte_auflisten.xml` dieser Activity. Wir binden es in der `onCreate`-Methode ein.

Listing 6-1 *Layout für die Geokontakt-Liste*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <Spinner
    android:id="@+id/sp_sortierung"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawSelectorOnTop="true"
    android:entries="@array/Sortierung"/> (1)

  <ListView (3)
    android:id="@+id/android:list"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textFilterEnabled="true"
    android:cacheColorHint="@color/hintergrund"/>

  <TextView
    android:id="@+id/android:empty"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=
      "@string/txt_geokontakt_auflisten_keineDaten"/> (2)
</LinearLayout>
```

Einen `Spinner` verwenden

Das Layout enthält als Erstes einen `Spinner`. Spinner sind Drop-down-Listen. Meist wird ein Spinner in einer Anwendung immer eine feste Liste von Werten haben. Diese Werte legt man in einer Array-Ressource ab. Wir legen dazu die Datei `/res/values/arrays.xml` an und füllen sie mit dem folgenden XML:

```
<resources>
  <string-array name="Sortierung">
    <item>Standard</item>
    <item>Name</item>
  </string-array>
</resources>
```

Das XML-Attribut `android:entries="@array/Sortierung"` (1) in der Spinner-Definition in Listing 6-1 lädt automatisch die Daten aus der Ressource und zeigt sie an.

Wir dürfen nicht vergessen, die Text-Ressource `txt_geokontakt_auflisten_keineDaten` der Datei `strings.xml` hinzuzufügen. Der Text kommt immer zur Anzeige, wenn die Liste der Geokontakte (3) leer ist. Als Text kann man »Keine Geokontakte vorhanden« verwenden.

6.3 AdapterViews und Adapter

Views mit Adaptern befüllen

Manche View-Elemente dienen der Anzeige von vielen Datensätzen. Sie werden als `AdapterView` bezeichnet und sind Views, deren Kindelemente durch Adapter mit Daten befüllt werden. Diese View-Elemente sind von `android.widget.AdapterView<T extends android.widget.Adapter>` abgeleitet, die wiederrum von `android.view.ViewGroup` abgeleitet ist. In Android sind folgende Views von AdapterView abgeleitet:

- `ListView`
- `Gallery`
- `GridView`
- `Spinner`

Den Spinner haben wir gerade in Listing 6-1 kennengelernt. Mit seiner Hilfe kann der Anwender die Sortierung ändern. Zur Anzeige der Geokontakte als Liste verwenden wir eine `ListView` (3).

Aufgaben des Adapters

Als Bindeglied zwischen einer Datenmenge und einer `AdapterView` dienen *Adapter*. Ein Adapter erfüllt zwei Aufgaben:

- Er füllt die `AdapterView` mit Daten, indem er ihr eine Datenquelle liefert.
- Er definiert, welche View bzw. Viewgroup zur Darstellung der einzelnen Elemente der Menge verwendet wird.

Wahl der Adapter

Anhand der Art und Weise, wie die Datenmenge definiert ist, hat man die Wahl zwischen verschiedenen Implementierungen des Interface `android.widget.Adapter`. Wir stellen hier den `ArrayAdapter` vor. Später

lernen wir noch den `SimpleCursorAdapter` kennen, der Daten aus einer Datenbank als Datenquelle nutzt. Mit Datenbanken beschäftigen wir uns intensiv in Kapitel 11. Alle anderen Adapter sind Variationen dieser beiden Adapter.

6.3.1 ArrayAdapter

In diesem Kapitel werden wir unsere Daten in Arrays speichern, daher stellen wir in Listing 6-2 den `ArrayAdapter` vor.

Adapter Schritt für Schritt

Listing 6-2 zeigt den Quellcode der ersten Version der Activity `GeoKontakteAuflisten`.

Listing 6-2 *Nutzung eines `ArrayAdapter`*

```
public class GeoKontakteAuflisten extends ListActivity {

  private String[] NAMEN = new String[5];

  private void initialisiereNamen() { // (1)
    NAMEN[0] = "Berthold Schmitz";
    NAMEN[1] = "Chantal Schulze";
    NAMEN[2] = "Bartolomäus Weissenbaum";
    NAMEN[3] = "Jean-Paul Küppers";
    NAMEN[4] = "Anneliese Müller";
  }

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.geokontakte_auflisten); // (2)

    zeigeGeokontakte(); // (3)
  }

  private void zeigeGeokontakte() {
    initialisiereNamen();
    mKontaktAdapter =
      new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, NAMEN); // (4)
    setListAdapter(mKontaktAdapter); // (5)
  }
}
```

`ListActivity`

Wenn eine Bildschirmseite hauptsächlich zur Darstellung einer Liste von Daten benötigt wird, sollte man eine `android.app.ListActivity` als Basis verwenden. Diese erweitert die Funktionalität einer normalen `Activity` durch folgende Punkte:

Layout serienmäßig Die `ListActivity` verfügt über eine implizite `android.widget.ListView` als Wurzel ihrer Bildschirmseite. Eine eigene Layoutdefinition ist möglich, aber nicht unbedingt notwendig.

Vordefinierte Callbacks Die Callback-Methoden zur Behandlung typischer Ereignisse für Listen (z. B. ein Datenelement wird ausgewählt) werden bereits von der Activity implementiert und können nach Bedarf überschrieben werden. Es müssen keine separaten Event-Handler deklariert werden.

Hilfestellung für Listenzugriffe Jede `ListActivity` bietet Methoden, mit deren Hilfe Informationen über die aktuelle Listenposition oder das aktuell ausgewählte Listenelement abgefragt werden können.

Implizites Layout

Man sollte von dem vordefinierten Layout einer `ListActivity` nur in Ausnahmefällen abweichen. Es sorgt dafür, dass die Listenelemente optimal angezeigt werden. Bei Bedarf wird eine vertikale Bildlaufleiste (Scollbar) automatisch ergänzt.

Adapter sind das Bindeglied zwischen Views und Daten.

Wir haben einige Namen fest im Programmcode in einem Array definiert (1). In der `onCreate`-Methode wird die `AdapterView` ermittelt (2). Danach werden die anzuzeigenden Daten geladen (3). Schließlich verbindet der Adapter die View mit den Daten und gibt noch den Ressourcenschlüssel des Layouts mit, das die einzelnen Einträge in der Liste formatiert (4). In unserem Fall verwenden wir ein Layout für einen solchen Listeneintrag, den das Android-SDK mitbringt (/codeandroid.R.layout.simple_list_item_1). Dieses Layout enthält lediglich eine `TextView`, die in der Liste einen einzelnen Namen anzeigt.

Android bringt noch weitere vorgefertigte Layouts für einzelne Zeilen in Listendarstellungen mit (Tabelle 6-1). Durch einen Blick in die Klasse `android.R.layout` kann man sehen, welche es gibt. Bei Bedarf können auch eigene Layoutdefinitionen verwendet werden (s. Online-Dokumentation), was optisch wesentlich ansprechendere Layouts für Listeneinträge erlaubt.

Tab. 6-1
Vorgefertigte Layouts für ListView

android.R.layout.	Beschreibung
simple_list_item_1	Einelementige Liste
simple_list_item_2	Zweielementige Liste
simple_list_item_checked	Liste mit Checkbox
simple_list_item_single_choice	Liste mit Einfachauswahl
simple_list_item_multiple_choice	Liste mit Mehrfachauswahl

Array-Ressourcen als Alternative

Im letzten Schritt wird der Adapter an die `AdapterView` übergeben und die Daten werden auf der Oberfläche angezeigt (5). Anstelle des fest im Programmcode implementierten Arrays könnte man auch eine Array-Ressource laden und die Werte dort ablegen. Falls man statische Listen im Programm verwendet, empfiehlt sich dies schon aus Gründen der Mehrsprachigkeit.

Abschließend muss die neue Activity noch im Android-Manifest bekannt gemacht werden. Hierzu fügen wir innerhalb des `application`-Tags die folgende Deklaration hinzu:

```
<activity android:name=".gui.GeoKontakteAuflisten" />
```

Um das Ergebnis zu testen, kann man in die Methode `onClick-GeokontakteVerwalten` der Activity `Startseite` folgenden Code zum Aufruf der `GeoKontakteAuflisten`-Activity einfügen.

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

Mit Intents zum Aufruf anderer Activities beschäftigen wir uns im Kapitel 7 näher. Nun bleibt noch die Frage, was passiert, wenn die Datenquelle keine Daten liefert. Dann ist die Liste leer und der Anwender sieht eine fast leere Seite. Besser ist es, einen Hinweistext anzuzeigen, dass die Liste leer ist. Dazu nutzen wir einen Automatismus der `ListView`. Das Layout der `GeoKontakteAuflisten`-Activity in Listing 6-1 enthält als letztes Element eine `TextView`, die immer nur dann angezeigt wird, wenn die Liste leer ist. Dies wird durch die Android-Id `android:id="@+id/android:empty"` erreicht. Der in dieser `TextView` hinterlegte Text wird immer dann angezeigt, wenn die Liste leer ist.

6.3.2 Auf Ereignisse reagieren

Wo bin ich?

Nun können wir Massendaten in Views anzeigen. Doch wie reagieren wir auf eine Auswahl aus einer Liste oder finden das aktuell markierte Element? Jede `AdapterView` erlaubt den Zugriff auf die Elemente seiner Datenmenge (`getItemAtPosition(int)`). Die Methoden `getSelectedItem` und `getSelectedItemPosition` liefern Informationen über das aktuell markierte Element der Datenmenge.

Reaktion auf Interaktion

Eine weitere Aufgabe einer `AdapterView` ist es, auf Nutzereingaben, die in ihren Anzeigebereich fallen, zu reagieren. Sie tut dies, indem sie je nach Aktion (Einfachklick, langer Klick etc.) des Nutzers ein Ereignis auslöst, auf das die Anwendung reagieren kann. Damit wären wir beim Thema des nächsten Abschnitts.

Auf Spinnerauswahl reagieren

Bei `AdapterViews` ist eine Auswahl durch Anklicken möglich. In einer Drop-down-Box (`Spinner`) oder einer `ListView` kann der Anwender einen Eintrag auswählen. Auf solch ein Oberflächenereignis muss im Programmcode reagiert werden. Im letzten Kapitel haben wir uns mit den Methoden befasst, die nach einem Klick auf ein Element eines Kontext- oder Optionsmenüs aufgerufen werden, um auf das Ereignis zu reagieren. Diese Methoden bezeichnen wir als *Callback-Methoden*.

Callbacks

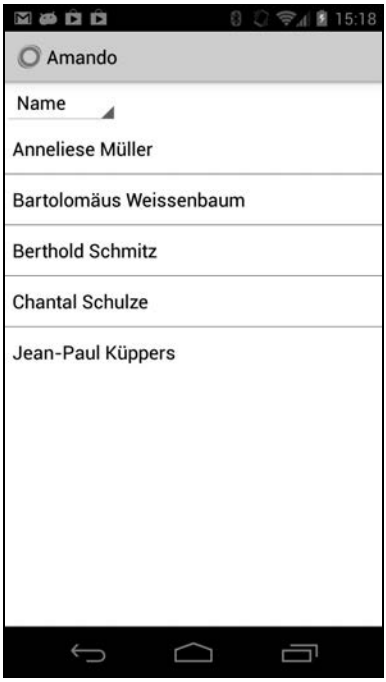

Abb. 6-1
Geokontakte sortieren

Die `AdapterViews` besitzen eigene Event-Handler-Klassen zur Behandlung solcher Auswahlereignisse. Es handelt sich wie bei Views (siehe Tabelle 5-21 auf Seite 81) um Interfaces für Event-Handler (Listener), die im Programmcode implementiert werden müssen. Tabelle 6-2 zeigt die möglichen Handler zum Reagieren auf Auswahlereignisse.

Listenauswahl behandeln

Implementieren wir zunächst den Event-Handler für den Spinner mit der Callback-Methode `onItemSelected` für den Fall, dass ein Element im Spinner ausgewählt wurde (Listing 6-3). Abbildung 6-1 zeigt die Activity `GeoKontakteAuflisten` mit der Auswahl, die der Spinner zur Verfügung stellt. Mittels des Spinners kann man auswählen, ob die Geodaten nach Namen sortiert oder nach Änderungsdatum angezeigt werden sollen.

Tab. 6-2 *Einige Event-Handler der Android-API*

Event-Handler	wird aktiviert, wenn…
AdapterView.OnItemClickListener	ein Datenelement kurz angeklickt wird
AdapterView.OnItemLongClickListener	ein Datenelement für längere Zeit angeklickt wird
AdapterView.OnItemSelectedListener	ein Datenelement ausgewählt wird

Listing 6-3 *Einen Listener für den Spinner implementieren*

```
private AdapterView.OnItemSelectedListener
    mSpinnerItemAuswahlListener =
        new AdapterView.OnItemSelectedListener() {
      @Override
      public void onItemSelected(AdapterView<?> arg0,
          View arg1, int position, long id) {

        switch (position) { // (1)
          case 0: // Standard
            initialisiereNamen(); // (4)
            mKontaktAdapter.notifyDataSetChanged();
          break;
          case 1: // Name // (2)
            Arrays.sort(NAMEN);
            mKontaktAdapter.notifyDataSetChanged(); // (3)
          break;
          default:
            // Spinner-Eintrag existiert nicht
          break;
        }
      }

      @Override
      public void onNothingSelected(AdapterView<?> arg0) {
      }
    };
```

Wir erschaffen uns ein Exemplar der Klasse `AdapterView.OnItemSelectedListener` und überschreiben die Methode `onItemSelected` (Listing 6-3). Als Parameter erhalten wir die Position des gewählten Elements im Spinner. Anhand dieser Position starten wir die gewünschte Aktion durch Verwendung einer Switch-Case-Anweisung.

Möglich ist eine Sortierung der Namen im Array `NAMEN` in alphabetischer Reihenfolge. Bei der Wahl von »Standard« wird hingegen wieder die ursprüngliche Reihenfolge angezeigt. Entsprechen sortieren wir das Array nach den Namen (2) mit Hilfe der statischen Methode

sort der Klasse Arrays. Nach der Sortierung wird der Adapter mittels mKontaktAdapter.notifyDataSetChanged (3) darüber informiert, dass sich die Daten in der Datenquelle geändert haben. Hier ist es wichtig zu wissen, dass das Datenobjekt zwar verändert, aber nicht ausgetauscht werden kann. Der Adapter hält eine Objektreferenz auf das Array NAMEN. Dieses darf daher nicht einfach mit new neu erzeugt und mit anderen daten befüllt werden. Dies ist auch der Grund, warum beim Wiederherstellen der ursprünglichen Reihenfolge das Array nur mit neuen Werten befüllt wird, ohne es neu zu erzeugen (4).

Nun fügen wir mSpinnerItemAuswahlListener noch den Spinner hinzu. Dies erfolgt in der onCreate-Methode der Activity. Dazu holen wir uns die AdapterView, also unseren Spinner, mittels der Ressourcenklasse R aus dem Layout und nutzen die Methode setOnItemSelectedListener (siehe Tabelle 6-2).

```
((Spinner) this.findViewById(R.id.sp_sortierung)).
    setOnItemSelectedListener(
        mSpinnerItemAuswahlListener);
```

Auf Listenauswahl reagieren

Nun haben wir gezeigt, wie man in der Activity auf die Auswahl in einem Spinner reagiert. Schauen wir uns nun an, wie wir auf die Auswahl eines Elements in der Liste der Geokontakte reagieren können.

Listing 6-4
Einen Listener für die ListView implementieren

```
@Override
protected void onListItemClick(ListView l, View v,
    int position, long id) {
  super.onListItemClick(l, v, position, id);

  final Toast hinweis = Toast
      .makeText(this, "Element "
          + ((TextView) v).getText(),
          Toast.LENGTH_LONG);
  hinweis.show();
}
```

Listing 6-4 zeigt den Vorteil der ListActivity. Sie besitzt die Methode onListItemClick, welche wir überschreiben, um einen Geokontakt aufzurufen. Der Listener, der beim Spinner zusätzlich implementiert werden musste, wird von ListActivity intern implementiert und wir haben alle notwendigen Parameter in der Methode direkt zur Verfügung. Eigentlich würden wir hier mittels eines Intents eine Activity zur Anzeige und Bearbeitung eines Geokontakts aufrufen. Mit Intents befassen wir uns jedoch erst in Kapitel 7. Daher lassen wir uns den ausge-

Listener muss nicht implementiert werden.

Kurznachricht: Toast

wählten Namen durch einen `android.widget.Toast` anzeigen. Toast sind kleine Meldungsfenster, die nach kurzer Zeit wieder verschwinden.

6.4 Anwendungseinstellungen

Konfiguration

Viele Anwendungen benötigen anwender- oder gerätespezifische Konfigurationen. Diese müssen während einer Anwendungssitzung angezeigt und bearbeitet werden können. Nach Beendigung der Anwendung darf deren Konfiguration nicht gelöscht werden. Sie muss beim nächsten Start unverändert wieder zur Verfügung stehen.

Noch keine Datenbanken

Die Android-API unterstützt uns bei der Verwaltung und Darstellung dieser Konfigurationseinstellungen, ohne dass dazu Datenbankkenntnisse notwendig sind. Sie stellt spezielle Oberflächenelemente und eine eigene Activity bereit, um die Darstellung von Einstellungen zu vereinfachen. Diese Komponenten sowie die zur Verwaltung der Konfigurationsdaten empfohlenen Klassen werden wir in diesem Abschnitt kennenlernen.

Amando-Konfigu ation

Bei Amando kann man sich selbst einen Nickname vergeben, der mit der Position an einen Bekannten gesendet wird. Zusätzlich kann man angeben, ob man gegen eine lokale Installation des Amando-Servers testet (was nur mit dem Emulator funktioniert), oder ob man die Serverinstallation im Internet verwenden möchte. Als letzte Kategorie von Einstellungen kann man noch die Häufigkeit der Positionsübermittlung einstellen. Die Amando-Anwendung erhält nur alle x Sekunden neue Positionsdaten vom GPS-Modul des Geräts (»Positionsermittlung (Zeit)«) oder wenn sich die Position seit der letzten Positionsermittlung um mehr als y Meter geändert hat (»Positionsermittlung (Distanz)«). Abbildung 6-2 zeigt die Einstellungsmöglichkeiten von Amando.

Die Einstellungsparameter sollten nicht als Java-Code oder Ressourcen definiert werden. Sie werden durch Anwender zur Laufzeit des Programms eingestellt und müssen gespeichert werden, um beim nächten Programmstart wieder zur Verfügung zu stehen.

Ziel: Einstellungen bearbeiten

Unser Ziel ist es, zur Verwaltung der Anwendungseinstellungen von Amando eine Bildschirmseite zu erstellen, die an das Hauptmenü angebunden werden kann.