

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 mit einem Cache-Mechanismus 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 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="fill_parent"
    android:layout_height="fill_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 (2)
        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"/>
</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:

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

6.3 AdapterViews und Adapter

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<T>` abgeleitet, die wiederum von `android.view.ViewGroup` abgeleitet ist. In Android sind folgende Views von `AdapterView` abgeleitet:

*Views mit Adaptern
befüllen*

- `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` (2).

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

Aufgaben des Adapters

- 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.

Anhand der Art und Weise, wie die Datenmenge definiert ist, hat man die Wahl zwischen verschiedenen Implementierungen des Interface `android.widget.Adapter`. Wir werden zwei Adapter vorstellen, den `ArrayAdapter` und den `SimpleCursorAdapter`. Alle anderen Adapter sind Variationen dieser beiden Adapter. Der `ArrayAdapter` operiert, wie der Name schon sagt, auf Arrays. Der `SimpleCursorAdapter` ist von `CursorAdapter` abgeleitet und verlangt einen `Cursor` als Datenquelle. Meist wird dies ein Datenbank-Cursor sein, mit dem wir einzelne Zeilen aus dem Ergebnis einer Datenbankabfrage erhalten. Mit Datenbanken beschäftigen wir uns intensiv in Kapitel 11. Das Verständnis des

Wahl der Adapter

Cursors ist hier noch nicht wichtig. Wissen muss man aber, dass der CursorAdapter in jedem Fall eine Spalte names `_id` erwartet.

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 static final String[] NAMEN =
        new String[] { // (1)
            "Berthold Schmitz",
            "Chantal Schulze",
            "Bartolomäus Weissenbaum",
            "Jean-Paul Küppers" };

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

        zeigeGeokontakte(); // (3)
    }

    /**
     * Zeige die Liste der GeoKontakte an.
     */
    private void zeigeGeokontakte() {
        final ArrayAdapter<String> kontaktAdapter =
            new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, NAMEN); // (4)
        setListAdapter(kontaktAdapter); // (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 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 des Anzeige-Cursors oder das aktuell ausgewählte Listenelement abgefragt werden können.

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.

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 Array-Elemente formatiert (4). Android bringt einige 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).

Implizites Layout

Adapter sind das Bindeglied zwischen Views und Daten.

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

Tab. 6-1
Vorgefertigte Layouts für ListView

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.

Array-Ressourcen als Alternative

6.3.2 SimpleCursorAdapter

Wenden wir uns nun komplexeren Datenquellen zu. Gehen wir davon aus, dass wir schon eine Datenbank mit den Geokontakten haben und eine Klasse `GeoKontaktColumns`, die die Spaltennamen der Tabellen definiert. Die Implementierung von `GeoKontaktColumns` und der Methode `ladeGeoKontaktListe` (1) erfolgt später in Kapitel 11 über Datenbanken. Zugriff auf die Ergebnismenge einer Datenbankabfrage erhält man über einen `Cursor`. Wir ändern die Methode `zeigeGeokontakte` und bringen die Werte, die uns der `Cursor` liefert, mittels eines `SimpleCursorAdapters` zur Anzeige (siehe Listing 6.3).

Cursor als Datenquelle

Listing 6.3
Nutzung eines
`SimpleCursorAdapter`

```
private static final String[] ANZEIGE_KONTAKTE =
    new String[] {
        GeoKontaktColumns.NAME,
        GeoKontaktColumns.STICHWORT_POS};

private static final int[] SIMPLE_LIST_VIEW_IDS =
    new int[] {
        android.R.id.text1, // (3)
        android.R.id.text2 };

private void zeigeGeokontakte() {
    final Cursor mcKontakt = mKontaktSpeicher. // (1)
        ladeGeoKontaktListe(null);
    startManagingCursor(mcKontakt);

    final SimpleCursorAdapter kontaktAdapter =
        new SimpleCursorAdapter(this,
            android.R.layout.simple_list_item_2, mcKontakt, // (2)
            ANZEIGE_KONTAKTE, SIMPLE_LIST_VIEW_IDS);
    setListAdapter(kontaktAdapter);
}
```

Für den `SimpleCursorAdapter` brauchen wir noch zwei Deklarationen, die wir der `Activity` hinzufügen. Dies sind zum einen die Spaltennamen der Ergebnismenge, die uns der `Cursor` liefert (`ANZEIGE_KONTAKTE`). Das ist nötig, da die Ergebnismenge noch wesentlich mehr Spalten enthalten könnte und wir daher angeben müssen, welche Spalten zur Anzeige kommen sollen. Zum anderen sind dies die Ressourcen-Ids des Layouts, welches wir zur Anzeige einer Ergebniszeile in der `ListView` verwenden (`SIMPLE_LIST_VIEW_IDS`). Wir verwenden ein `simple_list_item_2`-Layout (2), welches zwei `TextViews` enthält, die über die Ressourcen-Ids in `SIMPLE_LIST_VIEW_IDS` referenziert werden können (3).

Spaltennamen ...

*... und Ressourcen-Ids
angeben*

Der Adapter erhält alle notwendigen Attribute: den `Context`, das `Layout` für die Darstellung, die Daten mittels des `Cursors` und die Be-

schreibung, welche Spalten in welcher View zur Anzeige kommen sollen.

Falls die Datenmenge leer ist, wird die View standardmäßig ohne Werte angezeigt. Es ist jedoch möglich, mittels `AdapterView::setEmptyView` eine Referenz auf eine View zu übergeben, die in diesem Fall als Platzhalter angezeigt wird. Auf diese Weise ließe sich beispielsweise eine `TextView` mit einem Hinweis darstellen.

Leere Datenmenge

```
TextView hinweisKeinKontakt = new TextView(this);
hinweisKeinKontakt.setText(R.string.kein_kontakt_vorhanden);
getListView().setEmptyView(hinweisKeinKontakt);
```

6.3.3 Auf Ereignisse reagieren

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.

Wo bin ich?

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.

Reaktion auf Interaktion

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

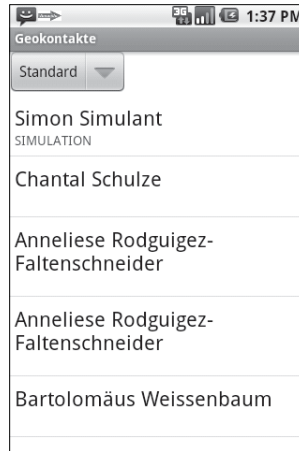
Die `AdapterViews` besitzen eigene Event-Handler-Klassen zur Behandlung solcher Auswahlereignisse. Es handelt sich wie bei Views (siehe Tabelle 5-22 auf Seite 71) um Interfaces für Event-Handler (Listener), die im Programmcode implementiert werden müssen. Tabelle 6-2 zeigt die möglichen Handler zum Reagieren 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 6.4). Abbildung 6-1 zeigt die Activity `GeoKontakteAuflisten` mit der Auswahl, die der Spinner zur

Abb. 6-1

Geokontakte sortieren

**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

Verfügung stellt. Mittels des Spinners kann man auswählen, ob die Geodaten nach Namen sortiert oder nach Änderungsdatum angezeigt werden sollen.

Listing 6.4Einen Listener für den
Spinner
implementieren

```
private AdapterView.OnItemSelectedListener
    mSpinnerItemAuswahlListener =
        new AdapterView.OnItemSelectedListener() {

    @Override
    public void onItemSelected(AdapterView<?> arg0,
        View arg1, int position, long id) {

        Cursor sortedCursor;
        switch (position) { // (1)
            case 0:
                sortedCursor = mKontaktSpeicher.
                    ladeGeoKontaktListe(Sortierung.STANDARD, null);
                break;
            case 1:
                sortedCursor = mKontaktSpeicher.
```

```

        ladeGeoKontaktListe(Sortierung.NAME, null);
        break;
    default:
        sortedCursor = mKontaktSpeicher.
            ladeGeoKontaktListe(null);
    }

    final ListView view = getListView(); // (2)
    final SimpleCursorAdapter cursorAdapter = // (3)
        ((SimpleCursorAdapter) view.getAdapter());

    cursorAdapter.changeCursor(sortedCursor); // (4)
}

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

}
};

```

Wir erschaffen uns ein Exemplar der Klasse `AdapterView.OnItemSelectedListener` und überschreiben die Methode `onItemSelected` (Listing 6.4). Als Parameter erhalten wir die Position des gewählten Elements im Spinner. Anhand dieser Position starten wir die gewünschte Aktion (1) durch Verwendung einer Switch-Case-Anweisung. Die Methode `ladeGeoKontaktListe` lädt alle Geokontakte neu aus der Datenbank, jedoch mit der gewünschten Sortierung, wenn der Parameter gesetzt ist. Abschließend wird die `ListView` ermittelt (2) und aus dieser der `SimpleCursorAdapter` (3). Die `ListView` können wir einfacher über die Methode `getListView` erhalten anstatt wie bisher über die Methode `findViewById`.

Zugriff auf ListViews

Nach der Sortierung steht er am Beginn der sortierten Ergebnismenge. Durch Austausch des Cursors (4) wird die Liste mit den Namen der Geokontakte neu aufgebaut. Für den Moment können wir alle unbekanntenen Programmteile auskommentieren, damit die Anwendung lauffähig bleibt.

Nun fügen wir `mSpinnerItemAuswahlListener` noch dem Spinner hinzu. 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 6.5

Einen Listener für die
ListView
implementieren

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

    // ToDo: rufe Anzeige Geokontakt auf.
    // Stattdessen:
    final Toast hinweis = Toast
        .makeText(this, "Element "
            + ((TextView) v).getText(),
            Toast.LENGTH_LONG);
    hinweis.show();
}
```

Listener muss nicht
implementiert werden.

Listing 6.5 zeigt den Vorteil der ListActivity. Sie besitzt die Methode onItemClick, 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 ausgewählten Namen durch einen android.widget.Toast anzeigen. Toast sind kleine Meldungsfenster, die nach kurzer Zeit wieder verschwinden.

Kurznachricht: Toast

6.4 Performante Listen

Langsame
Standardvariante

Wir haben oben den SimpleCursorAdapter kennengelernt, um eine Verbindung zwischen Datenquelle und View herzustellen. Die Standardimplementierung des SimpleCursorAdapter hat jedoch einen gravierenden Nachteil: Sie ist nicht performant.

Sichtbare Verzögerung

Wenn wir große Datenmengen haben, kommt dieser Nachteil zum Vorschein. Wenn wir eine lange Liste mit Einträgen anzeigen und uns in dieser schnell nach oben oder unten bewegen, so läuft diese Liste nicht gleichmäßig durch, sondern ruckelt zwischendurch. Android verwendet zwar für die View-Objekte, die die Listeneinträge repräsentieren, einen Cache, jedoch nicht für die Daten, die der Listeneintrag anzeigt.

Beim ersten Durchlaufen der Liste werden die View-Objekte erzeugt und später wiederverwendet. Die Daten werden aber jedesmal