

# Dauerhaftes Speichern von Daten (Persistenz)



# Inhaltsübersicht

2

1. Einführung in die Entwicklung mobiler Anwendungen
2. Erste grafische Oberflächen und Benutzerinteraktionen
3. Weiterführende Konzepte mobiler Plattformen
4. Standorbezogene Dienste, Sensoren und Kamera
5. Dauerhaftes Speichern von Daten (Persistenz)
6. Responsive Design, Weiterführende Interaktionsmuster
7. Asynchrone Verarbeitung



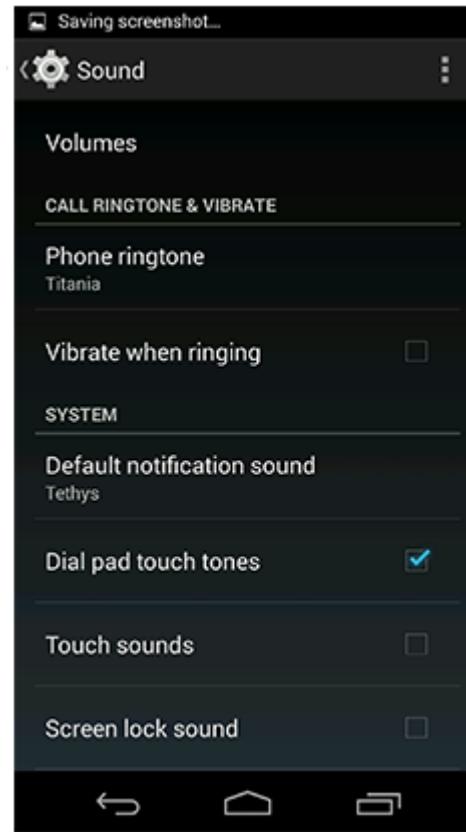
# Möglichkeiten der Persistierung in Android

3

- Shared Preferences
  - Jeder Anwendung steht ein einfacher Key-Value Store zur Verfügung, um primitive Datentypen (Boolean, Integer, Float, etc.) langfristig zu persistieren
- Interner Speicher
  - Jeder Anwendung steht eine interne Dateiablage zur Verfügung, die Dateien sind nur für die Anwendung sichtbar
- Externer Speicher (SD Karte)
  - Öffentliche Dateiablage auf dem externen Speicher
- Strukturierte Daten (SQLite)
  - Zugriff auf strukturierte Daten
- Content Provider u. Resolver
  - Standardisierter Zugriff auf Daten



- Anwendungen können über Shared Preferences primitive Datentypen (Boolean, Integer, Float, Long, String) persistieren
- Shared Preferences kommen hauptsächlich zum Einsatz um Benutzereinstellungen zu Persistieren, sind jedoch nicht darauf beschränkt



Speziell für die Verwaltung von Einstellungen gibt es die PreferenceActivity

# Shared Preferences Beispiel

- Abfragen von Shared Preferences

```
prefs = getSharedPreferences("MY_PREFERENCES", MODE_PRIVATE);  
  
String opt1 = prefs.getString("option1", "default");
```

Referenz auf das SharePreference Objekt, mit Bezeichner (MY\_PREFERENCES)

- Hinzufügen von Shared Preferences

Abfrage eines Strings (Wert) über den Schlüssel „option1“  
Falls noch kein Wert gesetzt wurde,  
wird der Defaultwert  
zurückgegeben (2. Parameter)

```
SharedPreferences.Editor editor = prefs.edit();  
editor.putString("option1", "option value");  
editor.apply();
```

Über den Editor können einfach weitere Schlüssel-Wert Paare hinzugefügt bzw. editiert werden

Nachdem alle Änderungen durchgeführt wurden, wird mit der Methode apply() der Stand persistiert

# Persistenz auf internem Gerätespeicher

6

- Anwendungen können beliebige Dateien auf dem internen Speicher des Geräts ablegen
- Diese Dateien können geschützt (privat) abgelegt werden, sodass nur die erstellende Anwendung direkten Zugriff hat
- Diese Dateien sind an den Lebenszyklus der Anwendung gebunden:
  - Falls die Anwendung deinstalliert wird, werden auch die Dateien gelöscht
- Der Dateipfad ist folgendermaßen aufgebaut:
  - /data/data/[Paketname der Anwendung]/...

# Speichern auf internem Gerätespeicher

7

```
String pers = input.getText().toString();

try {
    FileOutputStream fos = openFileOutput("output.txt", MODE_PRIVATE);
    fos.write(pers.getBytes());
    fos.close();
} catch (Exception ex) {
    Log.e(TAG, ex.getLocalizedMessage());
}
```

```
StringBuilder sb = new StringBuilder();

try {
    FileInputStream fis = openFileInput("output.txt");

    int len;
    byte[] buffer = new byte[256];
    while ((len = fis.read(buffer)) != -1) {
        sb.append(new String(buffer));
    }

    fis.close();
} catch (Exception ex) {
    Log.e(TAG, ex.getLocalizedMessage());
}
```

Die Methode **openFileOutput()** gibt die Referenz auf einen OutputStream im internen Gerätespeicher zurück

- MODE\_PRIVATE löscht eine bereits existierende Datei
- MODE\_APPEND fügt der Datei weitere Daten hinzu

Die Methode **openFileInput()** gibt die Referenz auf einen InputStream im internen Gerätespeicher zurück  
Byteweise können Strings aus dem Stream gelesen werden

# Persistenz auf externem Gerätespeicher

8

- Diese Methode bietet ähnliche Möglichkeiten wie das Speichern auf dem internen Gerätespeicher, jedoch mit einigen Ausnahmen
  - Alle Daten sind grundsätzlich öffentlich, andere Anwendungen haben Lese- bzw. Schreibzugriff darauf
  - Der Externe Speicher kann jederzeit von Benutzer aus dem Gerät entfernt werden bzw. beim Anschließen an einen Computer ausgeworfen werden (umount)
- (1) Die Daten können an den Lebenszyklus der Anwendung gekoppelt werden und werden mit Deinstallation der Anwendung ebenfalls entfernt
  - Dateipfad: /mnt/sdcard/Android/data/[Paketname der Anwendung]/...
- (2) Die Daten können in öffentliche Bereiche des externen Speichers abgelegt werden und bleiben nach der Deinstallation der Anwendung noch erhalten
  - Dateipfad: /mnt/sdcard/Android/...



- Der externe Speicher kann READ & WRITE Zugriff erlauben, nur READ oder keinen Zugriff

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

// Lesen und Schreiben auf dem External Storage möglich
if (Environment.MEDIA_MOUNTED.equals(state)) {
    mExternalStorageAvailable = mExternalStorageWriteable = true;
// nur lesend Zugriff
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```

Die Klasse Environment bietet eine statische Methode für die Abfrage des Zustands des Externen Speichers

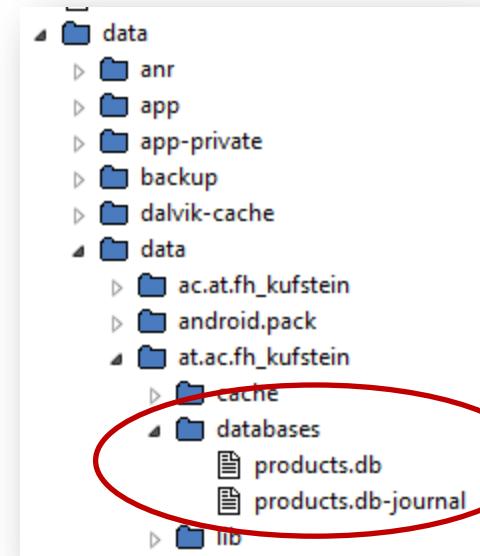
Abfragen über den Zustand des Externen Speichers:

- MOUNTED:  
Schreib- und Lesezugriff
- READ\_ONLY: nur Lesezugriff

# Benutzung von Datenbanken mit Android

10

- Jede App kann eine (SQL)-Datenbank für die Speicherung von Daten nutzen, die vom Android Framework bereitgestellt wird
  - Verwendet wird SQL-Lite
  - Jede Datenbank ist privat (nur für die jeweilige App nutzbar)
  - Wird gespeichert in /data/data/<package\_name>/database/



- Verfügbares Datenbankmanagementsystem: SQL-Lite
  - Open Source DB, standardkonform (SQL-92 mit Ausnahmen)
  - Leichtgewichtig und single-tier (einschichtig) als Bibliothek realisiert
  - Weitere Informationen: <http://www.sqlite.org/>

- Zugriff auf die Datenbank kann direkt oder über eine Hilfsklasse erfolgen:
  1. Direkter Zugriff auf das Datenbankobjekt über den Context:

```
SQLiteDatabase db = context.openOrCreateDatabase(DATABASE_NAME,  
                                              Context.MODE_PRIVATE,  
                                              null);
```

Zugriff auf die Datenbankfunktionalität über die Klasse SQLiteDatabase und die entsprechenden Methodenaufrufe (insb. execSQL())

Zugriff sollte möglichst sparsam eingesetzt werden, um Latenzen zu vermeiden – liegt im Ermessen des Programmierers

2. Indirekter Zugriff über die eine Hilfsklasse, die von SQLiteOpenHelper abgeleitet ist  
empfohlene BestPractice, weil über diese Klasse die Schreib/Lesezugriffe auf die DB durch die Verwendung von Caches optimiert werden



- Indirekter Zugriff abgeleitete Hilfsklasse `SQLiteOpenHelper`
  - Funktioniert als Facade (DesignPattern), das in diesem Fall zum einen den Zugriff auf die eigentliche Datenbank einfacher gestaltet und außerdem Cachingmethoden verwendet, um die Datenbanknutzung effizienter zu machen.
  - Eigene Klassen überschreiben üblicherweise die Methoden
    - `OnCreate()` – zum Erzeugen einer Datenbank und
    - `OnUpdate()` – zum Aktualisieren der Datenbank mit einer neuen Version

# Durchführen von Abfragen

13

- Abfragen erfolgen über die Methode query() des SQLiteDatabase-Objekts.
  - Dieses wird in einer lesenden oder schreibenden Version vom SQL Lite Open Helper instanziert. Für Abfragen .getReadableDatabase()
  - Query() enthält in den Parametern alle Komponenten einer SQL-Abfrage

```
String[] result_columns = new String[]{  
    ProductDBOpenHelper.PRODUCT_ID_COLUMN,  
    ProductDBOpenHelper.PRODUCT_NAME_COLUMN,  
    ProductDBOpenHelper.PRODUCT_PRICE_COLUMN};
```

Auswahl der Spalten in der Ergebnismenge (String-Feld)

```
String where = ProductDBOpenHelper.PRODUCT_ID_COLUMN + "=?";  
String whereArgs[] = new String[]{String.valueOf(currentProductId)};  
String groupBy = null;  
String having = null;  
String order = null;
```

Where-Klausel mit Parametern

Optional: Gruppierung, Aggregation, Sortierung

```
SQLiteDatabase db = dbHelper.getReadableDatabase();  
Cursor cursor = db.query(ProductDBOpenHelper.DATABASE_TABLE,  
    result_columns, where,  
    whereArgs, groupBy, having, order);
```

Parameter für Where

Eigentliche Abfrage, die einen Cursor liefert

# Auswertung eines Abfrageergebnisses

14

- Eine Abfrage liefert als Ergebnis ein Cursorobjekt (Cursor) zurück, das in der Ergebnismenge auf die aktuelle Zeile zeigt.
- Cursor bietet zahlreiche Methoden an um
  - Den Cursor innerhalb des Ergebnissets zu positionieren (z.B. `moveToNext()`)
  - Die Daten unter der aktuellen Cursorposition zu lesen (z.B. `getString()`)
- Achtung: Cursor muss nach dem Verlassen der Schleife positioniert werden, um Ressourcen freizugeben!

Positionierung auf die nächste Zeile

Auslesen der Zeilenwerte (indexbasiert)

```
StringBuilder sb = new StringBuilder();
while (cursor.moveToNext()) {
    String productName = cursor.getString(cursor.getColumnIndex(ProductDBOpenHelper.PRODUCT_NAME_COLUMN));
    float productPrice = cursor.getFloat(cursor.getColumnIndex(ProductDBOpenHelper.PRODUCT_PRICE_COLUMN));
    sb.append("Product:").append(productName)
      .append(" Price:").append(productPrice).append("\n");

    Log.v(LOG_LABEL, "Product:" + productName + " Price:" + productPrice);
}
```

# Einfügen von neuen Datensätzen

15

- Neue Daten werden mit Hilfe der ContentValues-Klasse erfasst
  - Enthält die Daten als Schlüssel/Wert-Paare, bestehend aus Attributname und Inhalt
  - Zuweisung mit Hilfe der Methode put()
- Schreiben der Daten auf eine schreibbare Version der Datenbank (getWritableDatabase()) mit Hilfe der insert-Methode

Erstellung der ContentValues-Instanz und Befüllen der Werte

```
ContentValues newValues = new ContentValues();
newValues.put(ProductDBOpenHelper.PRODUCT_NAME_COLUMN, productName);
newValues.put(ProductDBOpenHelper.PRODUCT_PRICE_COLUMN, productPrice);
SQLiteDatabase db = dbHelper.getWritableDatabase();
db.insert(ProductDBOpenHelper.DATABASE_TABLE, null, newValues);
```

Schreibbare Instanz  
der DB besorgen

Einfügen in die Datenbank

# Löschen von Datensätzen

16

- Löschen von Datensätzen erfolgt ebenfalls auf eine schreibbaren Version der Datenbank mit Hilfe der delete-Methode
  - Ohne where-Selektion werden alle Datensätze gelöscht
  - Mit Hilfe des Where-Parameters (String) und des WhereArgs-Parameters (String-Feld) können die zu löschen Sätze selektiert werden
  - Methode liefert int zurück – entspricht der Anzahl der gelöschten Sätze

Selektion der Datensätze über Where mit „?“-Parameter

```
String where = ProductDBOpenHelper.PRODUCT_ID_COLUMN + " = ?";  
String whereArgs[] = new String[]{String.valueOf(deleteId)};  
SQLiteDatabase db = dbHelper.getWritableDatabase();  
db.delete(ProductDBOpenHelper.DATABASE_TABLE, where, whereArgs);  
db.close();
```

Löschen der Daten

Setzen des  
Parameterwerts

# Ändern von Datensätzen

17

- Ändern von Datensätzen ist ähnlich dem Löschen, es kommt die update-Methode zum Einsatz
  - Die zu ändernden Attribute sind wie beim Insert über ein ContentValues-Objekt bereitzustellen
  - Mit Hilfe des Where-Parameters (String) und des WhereArgs-Parameters (String-Feld) können die zu ändernden Sätze selektiert werden
  - Methode liefert int zurück – entspricht der Anzahl der geänderten Sätze

Definition der zu ändernden Attribute und der neuen Werte

```
ContentValues updatedValues = new ContentValues();
updatedValues.put(ProductDBOpenHelper.PRODUCT_NAME_COLUMN, _name);
updatedValues.put(ProductDBOpenHelper.PRODUCT_PRICE_COLUMN, _price);

String where = ProductDBOpenHelper.PRODUCT_ID_COLUMN + "=?";
String whereArgs[] = new String[]{String.valueOf(currentProductId)};
SQLiteDatabase db = dbHelper.getWritableDatabase();
db.update(ProductDBOpenHelper.DATABASE_TABLE, updatedValues, where, whereArgs);
```

Aktualisieren der Daten

Dateneinheiten speichern von Daten (persistenz)  
Studiengang Web-Business & Technology, WS 2014/15

Selektion der  
Datensätze über  
parametrisiertes  
Where

Softwareentwicklung Mobile  
Stefan Huber

- Content Provider stellen ein Standard Interface zur Verfügung um das Bearbeiten von Daten unter Android zu vereinheitlichen
- Content Provider präsentieren die Daten für externe Anwendungen als Tabellen, Spalten u. Zeilen ähnlich wie in SQL Datenbanken, Content Provider müssen aber nicht mit SQL realisiert sein!
- Der Zugriff auf Content Provider wird über Content Resolver Objekte realisiert
- Das Erstellen eines Content Providers für eigene Anwendungen ist nur in wenigen Fällen sinnvoll:
  - Andere Anwendungen benötigen Zugriff auf die Daten
  - Einbindung der Daten in die Suchfunktion von Android
- Android stellt bereits einige Content Provider (Kontakte, Kalender, Wörterbuch, etc.) zur Verfügung, welche über Content Resolver abgefragt werden können

# Zugriff auf Daten mit Content Resolver

19

- Content Resolver bieten Methoden um alle CRUD-Operationen abzudecken:
  - Create: Neue Daten hinzufügen

final Uri insert(Uri url, ContentValues values)  
Inserts a row into a table at the given URL.

- Read: Daten abfragen

final Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)  
Query the given URI, returning a Cursor over the result set.

- Update: Daten aktualisieren

final int update(Uri uri, ContentValues values, String where, String[] selectionArgs)  
Update row(s) in a content URI.

- Delete: Daten löschen

final int delete(Uri url, String where, String[] selectionArgs)  
Deletes row(s) specified by a content URI.



# Content URIs zur Identifikation von Daten

20

content://provider\_name/table\_name/id

- Innerhalb des Android Frameworks werden URIs verwendet um Daten zu identifizieren
- Das **Schema** (content) gibt Aufschluss darüber, dass es sich um Daten handelt die von Content Providern verwaltet werden
- Die **Authority** stellt den symbolischen Namen des Content Providers dar
- Der **Pfad** gibt Aufschluss über den Tabellennamen
  - Weitere Pfadelemente können ID eines dedizierten Datensatzes bezeichnen



# Datenabfrage über den Content Resolver

21

- Analog zum Abfragen von Datensätzen aus SQLite Datenbanken bieten Content Resolver eine **query()** Methode und liefern als Ergebnis ein **Cursor** Objekt

```
public final Cursor query (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
```

Die URI gibt Aufschluss über den Content Provider bzw. die Tabelle (SQL from)

Die Spalten der Tabelle die Ausgewählt werden sollen, null liefert alle

Kriterien für das Auswählen von Datensätzen (WHERE), mit ? werden Argumente markiert

Die ? der Auswahl Kriterien werden mit den übergebenen Argumenten aufgefüllt

Angabe der Sortierung des Resultats

# Datenmanipulation über den Content Resolver

22

```
public final Uri insert (Uri url, ContentValues values)
```

Für das Einfügen von Daten muss einerseits die URI (Content Provider u. Tabelle) angegeben werden, andererseits ein ContentValues Objekt für die Schlüssel-Wert Paare die eingefügt werden sollen.

```
public final int update (Uri uri, ContentValues values, String where, String[] selectionArgs)
```

Für das Update wird ebenfalls die URI benötigt. Des Weiteren müssen die ContentValues mitgegeben werden die aktualisiert werden sollen und die Auswahlkriterien (WHERE u. Argumente) für die Datensätze die Aktualisiert werden sollen.

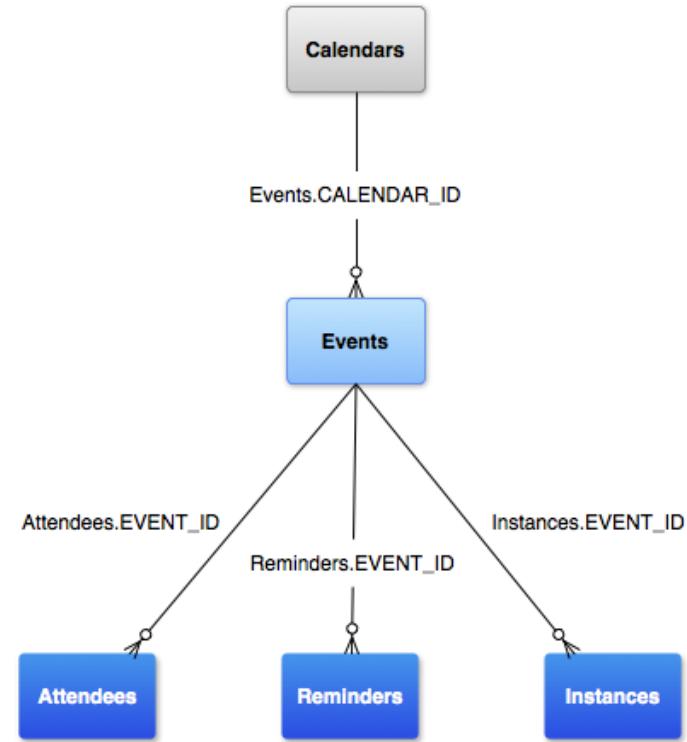
```
public final int delete (Uri url, String where, String[] selectionArgs)
```

Zum Löschen muss einerseits wiederum die URI angegeben werden und Auswahlkriterien für die Datensätze die gelöscht werden sollten.

# Beispiel: Arbeiten mit dem Kalender

23

- Android bieten einen Content Provider für das Arbeiten mit dem Kalender
- Der Kalender Content Provider verwendet intern eine SQLite Datenbank für das Ablegen von Daten
- Um Lese- bzw. Schreibzugriff auf den Kalender zu erhalten müssen entsprechende Permissions im Manifest gesetzt werden



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

Auswahl des Datums und der Uhrzeit des Termins.

```
Calendar c = Calendar.getInstance();
c.set(date.getYear(), date.getMonth(), date.getDayOfMonth(), time.getCurrentHour(), time.getCurrentMinute());

ContentResolver cr = getContentResolver();
ContentValues values = new ContentValues();
values.put(Events.DTSTART, c.getTimeInMillis());
values.put(Events.DTEND, c.getTimeInMillis() + 3600 * 1000);
values.put(Events.TITLE, edit.getText().toString());
values.put(Events.CALENDAR_ID, calID);
values.put(Events.EVENT_TIMEZONE, TimeZone.getDefault().getID());

Uri uri = cr.insert(Events.CONTENT_URI, values);
```

Referenz auf ein Content Resolver Objekt

Befüllen des ContentValues  
Objektes mit Schlüssel-Wert Paare,  
welche in die Tabelle eingetragen  
werden sollen

Einfügen des Termins. Als Ergebnis  
wird ein URI Objekt als  
Identifikator des Termins  
retourniert



# Kalender Beispiel: Abfragen von Terminen

25

Spezifikation der Selektionskriterien

Alle Termine die größer als heute sind

Erstellen des Projection Objekt, welches die Spalten spezifiziert die abgefragt werden sollen

```
ContentResolver cr = getContentResolver();

String[] projection = {Events.DTSTART,Events.DTEND,Events.TITLE};
String selection = Events.DTSTART + " > ?";
String[] selectionArgs = {Calendar.getInstance ().getTimeInMillis () + ""};

Cursor c = cr.query(Events.CONTENT_URI, projection, selection, selectionArgs, null);

while (c.moveToNext ()) {
    Log.i(TAG, c.getLong (0) + " " + c.getLong (1) + " " + c.getString (2));
}
```

Analog zur Abfrage von SQL Datensätzen, wird ebenfalls ein Cursor Objekt zur Verfügung gestellt, welches das Navigieren durch die Resultate ermöglicht

# Darstellung von Datenbankinhalten

26

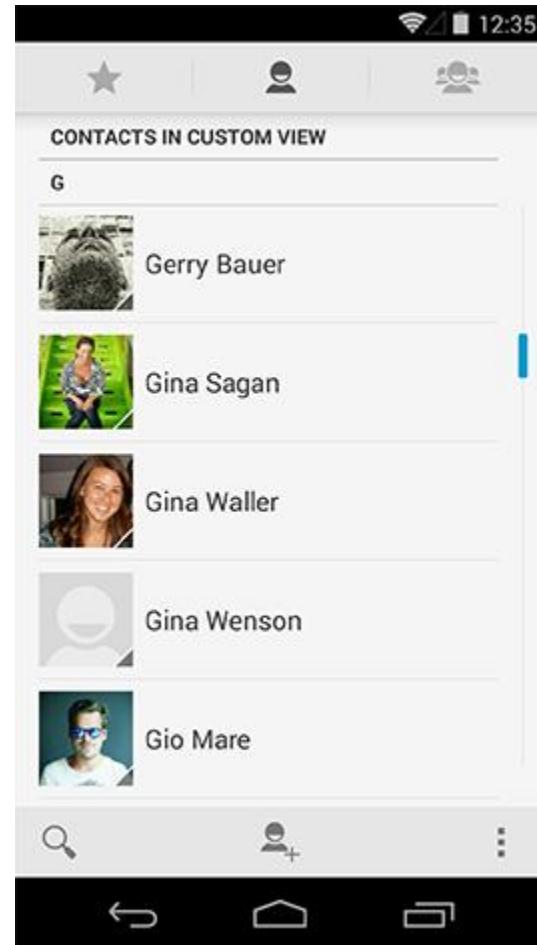
- Herausforderung: große Datenmengen
  - Speicher, Performanz
- Bekannter Ansatz aus der Übung: ScrollView
  - Alle Views (auch nicht sichtbare) müssen erzeugt werden.
  - Dies funktioniert nur bis zu einer gewissen Anzahl von Elementen
- Lösungsansatz: Adapter / AdapterView
  - Elemente werden on Demand geladen
  - Views werden wiederverwendet



# ScrollView vs AdapterView

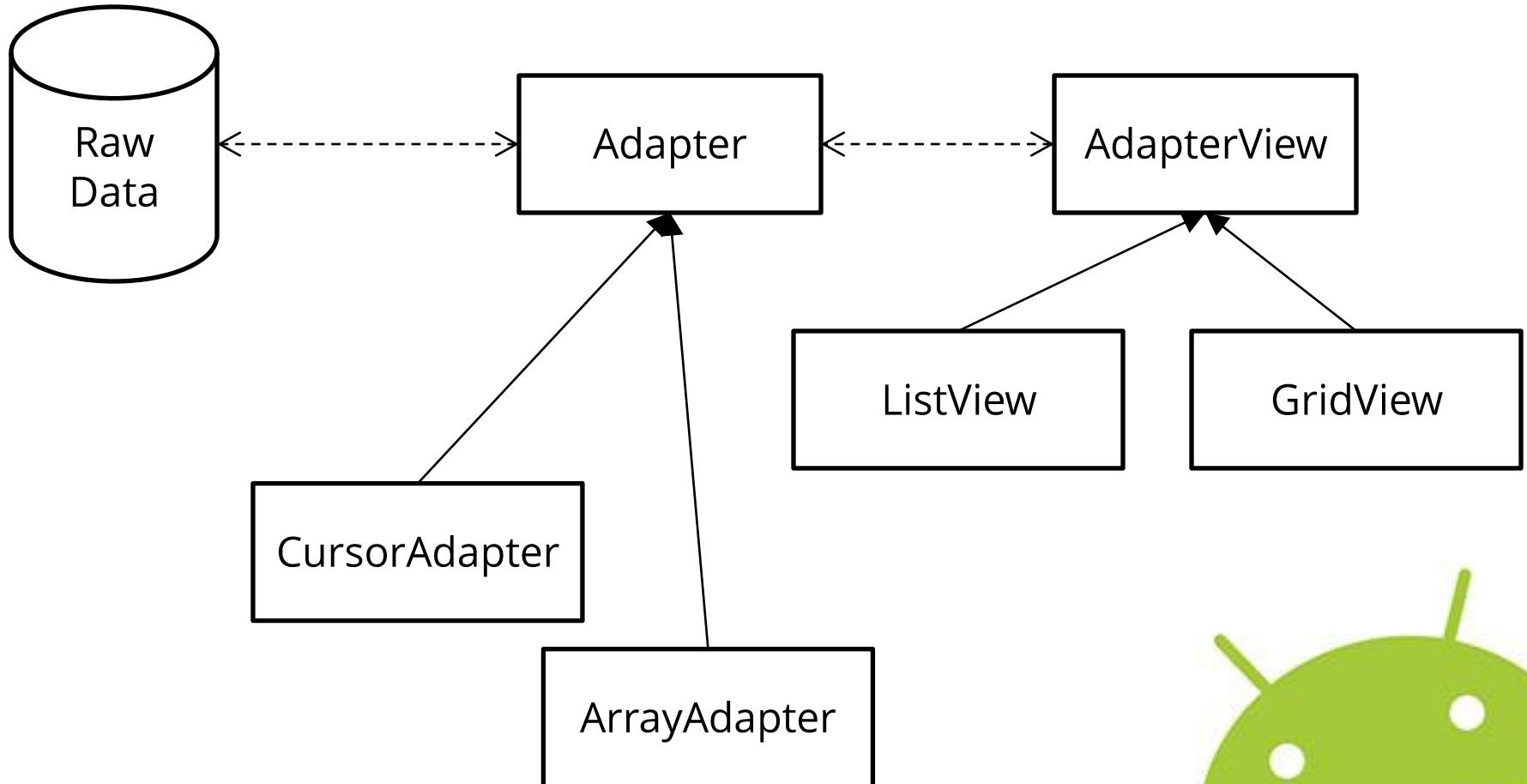
27

- ScrollView
  - alle Views im Speicher
- AdapterView
  - load on Demand
  - Views wieder-verwenden



# AdapterView und Adapter

28



# Beispiel SimpleCursorAdapter, ListActivity

29

Referenzen auf Spalten innerhalb der Abfrage

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,  
                      ContactsContract.CommonDataKinds.Phone.NUMBER};  
int[] toViews = {R.id.display_name, R.id.phone_number};
```

Referenzen auf IDs innerhalb des View Layouts

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,  
                    R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);  
ListView listView = getListView();  
listView.setAdapter(adapter);
```

Verknüpfung des  
Adapters mit der  
ListView

Angabe eines  
Layouts für einzelne  
Elemente

Angabe eines Cursors auf  
eine Abfrage



# „Take-Away“ für diese Einheit

30



- Verschiedene Ansätze zum Persistieren von Daten
- Arbeiten mit SQL Datenbanken auf mobilen Geräten
- Arbeiten mit Content Providern und Content Resolvern
- Darstellung von Datenbankabfragen