

Mobile Programming

Android 3 – Persistenz & Content Providers



kaspar.vongunten@hslu.ch



Inhalt

- Lokale Persistenz
 - Shared Preferences
 - Key/Value Storage, mit oder ohne UI
 - Dateisystem (intern + extern)
 - Binäre Dateien, Textdateien
 - Datenbank (Room DB)
 - Strukturierte Daten, Abfragen via ORM
- Content Providers
 - Daten teilen: z.B. Contacts, SMS, Kalender, ...
 - Intern oder mit anderen Apps
- Permissions
 - Aktionen, welche die Erlaubnis des Benutzers erfordern

Lokale Persistenz: 3 Möglichkeiten

■ Preferences

- Schlüssel/Werte-Paare (Key, Value)
- Verwendung für kleine Datenmengen

Persistenz = Daten über Laufzeit der App erhalten

Lokal, daher ist z.B. Web-Storage (Cloud, Backend, etc.) noch kein Thema. Backend-Kommunikation schauen wir später im Modul an...

■ Dateisystem, intern oder extern (SD-Karte)

- In App-Sandbox (privat) oder auf SD-Karte (öffentlich)
- Verwendung für binäre Daten, grosse Dateien, Export

■ Datenbank (Room)

- SQLite + Object Relational Mapper (ORM)
- Verwendung für strukturierte Daten + Abfragen/Suche



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Preferences

SharedPreferences

- Persistente Einstellungen für Activity oder Applikation
 - Key-Value-Store («persistente Map»)
 - Preferences für Activity:
 - `Activity.getSharedPreferences(mode)`
 - Anwendungsfall: Activity-State persistent speichern
 - Für Applikation:
 - `PreferenceManager.getDefaultSharedPreferences(ctx)`
 - `Context.getSharedPreferences(name, mode)`
- Mögliche Datentypen für Preferences-Werte
 - `String`, `int`, `long`, `float`, `boolean`
 - `Set<String>` (mit separaten Werten)

SharedPreferences: lesen & schreiben

- Mehrere Dateien pro Applikation sind möglich
 - Zugriff: `Activity.getSharedPreferences(name, mode)`

- Lesen

- Methoden `SharedPreferences.getX(...)`

Unterschiedliche
Datei-Namen

- Schreiben (immer mit Editor)

1. `editor = preferences.edit()`

2. `editor.putX(...)`

X = Typ, also z.B.
String, Int, Boolean, ...

3. `editor.apply()`

Damit werden
Änderungen persistiert

bevorzugt!

- Persistiert asynchron, d.h. nicht-blockierende Methode
- Falls synchron (blockierend) gewünscht: `editor.commit()`

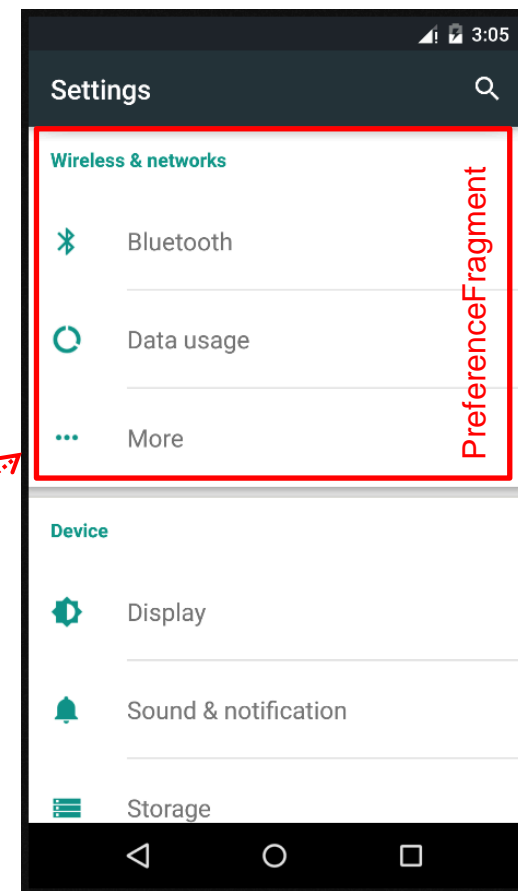
Demo: Zustand persistieren

- Wir persistieren Anzahl Aufrufe von onResume() über die Lebenszeit der App hinaus
- Anzeige auf Activity

```
final SharedPreferences preferences = getPreferences(MODE_PRIVATE);  
final int newResumeCount = preferences.getInt(COUNTER_KEY, 0) + 1;  
final SharedPreferences.Editor editor = preferences.edit();  
editor.putInt(COUNTER_KEY, newResumeCount);  
editor.apply();
```

User-Preferences: Darstellung

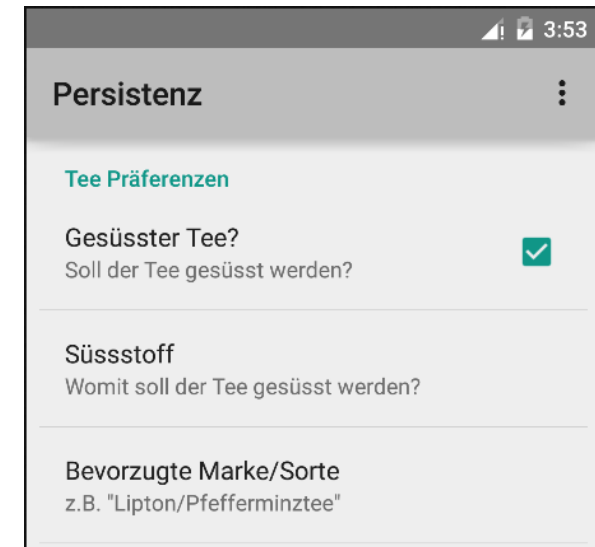
- Bekannt aus der Settings-App
 - Diesen Mechanismus können wir auch verwenden!
- Deklaration im XML = «Screen»
 - Darstellung „automatisch“ mit Hilfe von `PreferenceFragment`
 - Jeder Wertetyp hat eigenen Editor
- `PreferenceFragment` schreibt/liest grundsätzlich die `DefaultSharedPreferences`
 - Kann aber für anderen Preference-Store konfiguriert werden



User-Preferences: Beispiel „Tee Präferenzen“

- Benutzer soll angeben...
 - Gesüsst: Ja / Nein (boolean)
 - `CheckBoxPreference`
 - Süsstoff: Auswahl aus Liste (string-array)
 - `ListPreference`
 - Bevorzugte Marke: Freitext (string)
 - `EditTextPreference`

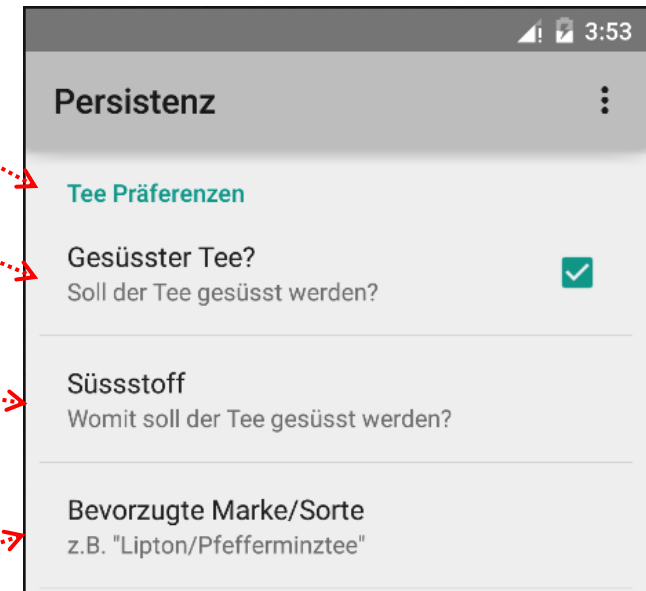
☞ siehe XML-Datei nächste Folie



User-Prefs: Deklaration

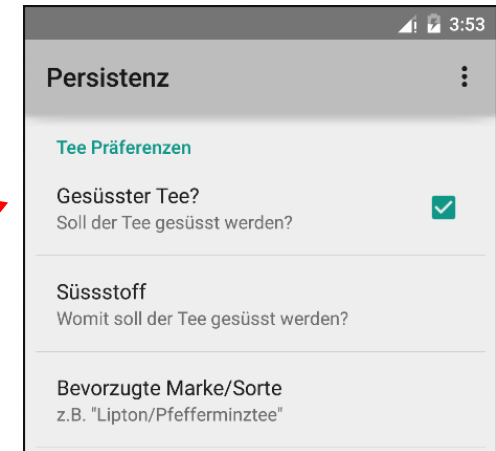
- Deklaration von User-Preferences in XML
 - Im Ordner `res/xml`, z.B. Datei `preferences.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
  <PreferenceCategory
    android:key="teaPrefs"
    android:title="Tee Präferenzen">
    <CheckBoxPreference
      android:key="teaWithSugar"
      android:persistent="true"
      android:summary="Soll der Tee gesüsst werden?"
      android:title="Gesüsster Tee?" />
    <ListPreference
      android:dependency="teaWithSugar"
      android:entries="@array/teaSweetener"
      android:entryValues="@array/teaSweetenerValues"
      android:key="teaSweetener"
      android:persistent="true"
      android:shouldDisableView="true"
      android:summary="Womit soll der Tee gesüsst werden?"
      android:title="Süsstoff" />
    <EditTextPreference
      android:key="teaPreferred"
      android:persistent="true"
      android:summary="z.B. "Lipton/Pfefferminztee""
      android:title="Bevorzugte Marke/Sorte" />
  </PreferenceCategory>
</PreferenceScreen>
```



User-Prefs: Erzeugung PreferenceFragment

- Im Beispiel als statische Klasse innerhalb der TeaPreferenceActivity:

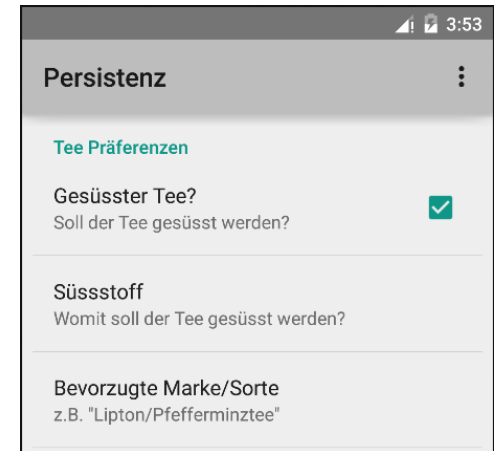


```
public static final class TeaPreferenceInitializer extends PreferenceFragment {  
    @Override  
    public void onCreate(final Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        addPreferencesFromResource(R.xml.preferences);  
    }  
}
```

Referenziert
res/xml/preferences.xml
aus voriger Folie

User-Prefs: Fragment anzeigen in Activity

- Fragment erzeugen und der Activity als Inhalt setzen



```
public class TeaPreferenceActivity extends Activity {  
  
    @Override  
    protected void onCreate(final Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        getSupportFragmentManager().beginTransaction().replace(android.R.id.content,  
            new TeaPreferenceInitializer()).commit();  
    }  
}
```

Mehr zu Fragmenten
später im Modul...

ID des Platzhalter-Elements
für einzufügendes Fragment

Hier: Root-Element

Demo: Preferences mit XML

- Activity für Tee-Präferenzen vom Benutzer:

- Boolean: „Mit Süsstoff?“

Editierbarkeit abhängig vom Wert von „Mit-Süsstoff“

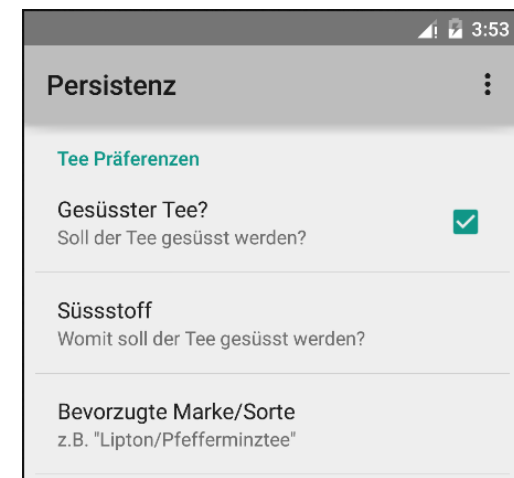
- List: „Süsstoff“

- (Assugrin, Kristallzucker, Rohrzucker)

- String: „Bevorzugte Marke/Sorte“

- Preference-Bildschirm definieren in xml

- `res/xml/preferences.xml`



Ausgegraut wegen
`android:dependency="teaWithSugar"`

```
<ListPreference  
    android:dependency="teaWithSugar"  
    android:entries="@array/teaSweetener"
```

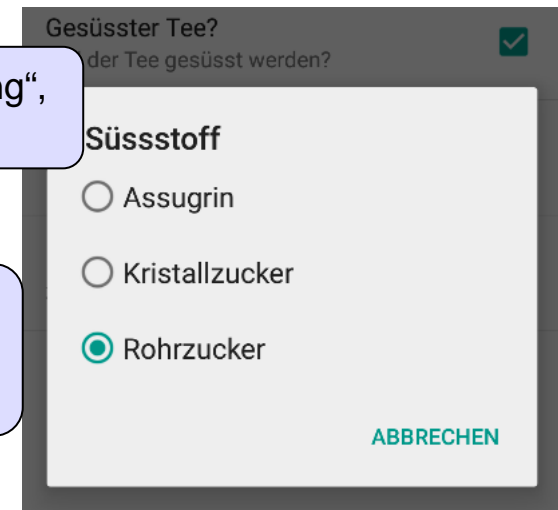
Hinweis: Daten für ListPreference aus Arrays

■ res/xml/preferences.xml

```
<ListPreference
    android:dependency="teaWithSugar"
    android:entries="@array/teaSweetener"
    android:entryValues="@array/teaSweetenerValues"
    android:key="teaSweetener"
    android:persistent="true"
    android:shouldDisableView="true"
    android:summary="Womit soll der Tee g
    android:title="Süsstoff" />
```

Entry = „Anzeigestring“,
Übersetzbar

EntryValue = „Werte“,
nicht übersetzt =
technischer Key



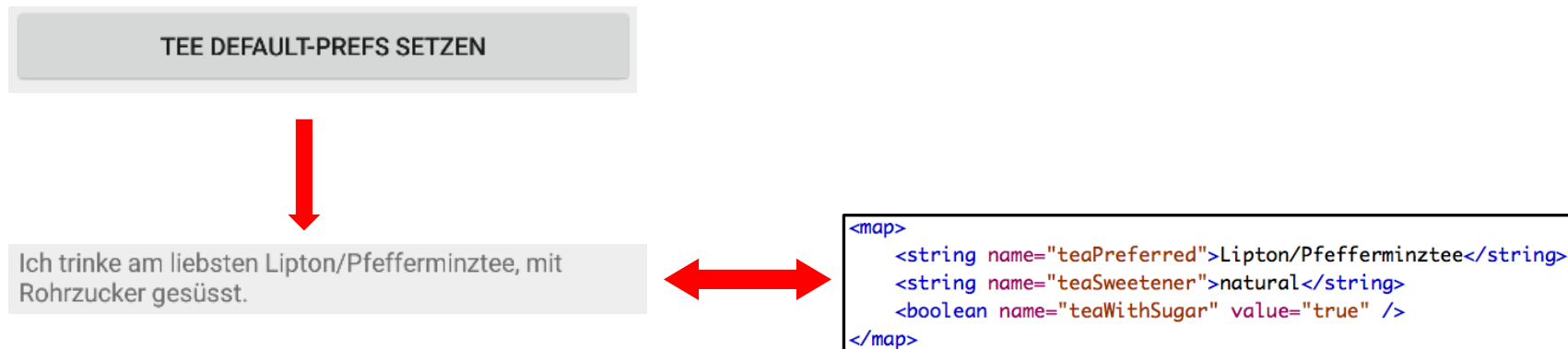
■ res/values/arrays.xml :

```
<resources>
    <string-array name="teaSweetenerValues">
        <item>artificial</item>
        <item>refined</item>
        <item>natural</item>
    </string-array>

    <string-array name="teaSweetener">
        <item>Assugrin</item>
        <item>Kristallzucker</item>
        <item>Rohrzucker</item>
    </string-array>
</resources>
```

Demo: Tee-Präferenz programmatisch setzen

- Bei Klick auf Button sollen Tee-Präferenzen programmatisch auf fixe Werte gesetzt werden
 - Verwendung Default-Preferences:
`PreferenceManager.getDefaultSharedPreferences(this)`
dann: `Editor editor = prefs.edit()`, usw. (d.h. fixe Werte für die drei Einstellungen setzen und speichern)





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Dateisystem

Dateisystem

- Einsatzbereiche
 - Speichern/Laden von binären Daten
 - Bilder, Musik, Video, Java-Objekte (Serialisierung), usw.
 - Caching
 - Heruntergeladene Dateien/Daten
 - Grosse Text-Dateien
 - Plain-Text
 - Strukturierte Daten (z.B. XML oder JSON)
- Teilen / Freigeben von erstelltem Inhalt
 - Externer Speicher (SD-Karte)

Zugriff Android Dateisystem

- Grundsätzliche Unterscheidung: Dateien sind...

Zugriff für andere Apps nur über Content Provider möglich

- PRIVATE → ins Applikationsverzeichnis

- `Context.getFilesDir()`

Ab KitKat (API 19) auch *privates* Verzeichnis auf SD-Karte möglich. Vor Gebrauch unbedingt Status des external Storage prüfen!

- PUBLIC → auf SD-Karte

- `Environment.getExternalStorageDirectory()`
`Environment.getExternalStorageState();`

- **Wichtig: Zugriff auf SD-Karte muss Erlaubnis beantragen!**
D.h. Eintrag notwendig im Manifest:

```
<manifest
  package="ch.hslu.mobpro.persistence"
  xmlns:android="http://schemas.android.com/apk/res/android">
```

Nicht nötig, falls im PRIVAT Modus geschrieben

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

Exkurs: Android Permission-Model



Permissions



- Gewisse Operationen von Apps benötigen eine Permission, die vom Benutzer erteilt werden muss
 - Zugriff auf Kontakte, Internet, SD-Karte, Kamera, SMS, Telefonieren, Apps deinstallieren, etc.
- Erforderliche Permissions werden von der App im Manifest deklariert

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    package="ch.hslu.mobpro.persistence"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.WRITE_SMS" />

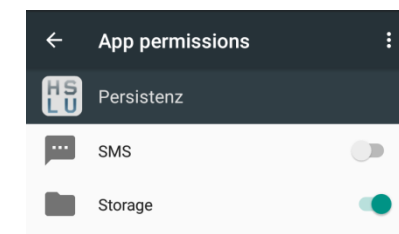
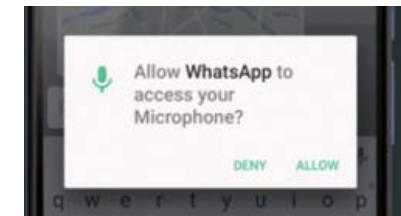
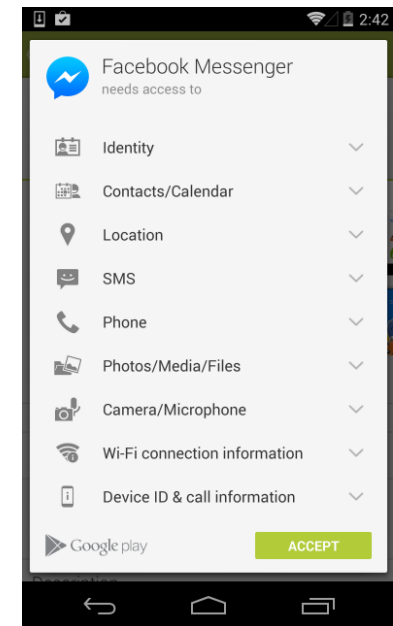
</manifest>
```

Normal,
Dangerous,
Signature,
System

- Klasse `android.Manifest.permission`
 - Auflistung aller Permissions, Groups, Protection Level

Permissions vor/nach Android 6

- In Android API < 23 (d.h. vor Android 6 / M) mussten alle Rechte vom Benutzer bei der Installation gewährt werden.
 - Alles oder nichts
- Seit API 23 werden bei der Installation keine 'dangerous' Permissions mehr gewährt (nur unkritische)
 - App muss für jede kritische Permission beim Benutzer nachfragen (wenn zum 1. Mal benötigt)
 - Permissions können einzeln abgelehnt oder entzogen werden (Settings > Apps > ... > Permissions)
 - Konsequenz: Apps müssen mit teilweise gewährten Permissions umgehen können!



Permissions in Android 6

■ Arten von Permissions

- normal, dangerous, signature, **signatureOrSystem**

Wird automatisch erlaubt für Apps, die im System-Image sind oder wie «signature».

Wird bei Installation automatisch erlaubt

Muss von User erlaubt werden (und kann wieder entzogen werden)

Wird automatisch erlaubt, wenn App, welche Permission definiert, von gleichem Hersteller, wie App, die Permission beanträgt. Sonst wie «dangerous».

Liste mit allen «normal» Permissions (u.a. **INTERNET**, BLUETOOTH, NFC, VIBRATE, ...):
<https://developer.android.com/guide/topics/permissions/normal-permissions.html>

■ Permissions können gruppiert werden

Warum das wohl «normal» ist?

- User gibt Freigabe für alle Permissions in einer Gruppe (nicht für Einzelpermission), falls benötigt

Runtime Permissions in Android 6 (API 23)



- Konzept: [.../guide/topics/security/permissions.html](http://developer.android.com/guide/topics/security/permissions.html)
- Howto: <http://developer.android.com/training/permissions/index.html>

```
public void loadExtFileWithPermission() {  
    int grant = checkSelfPermission(Manifest.permission.READ_EXTERNAL_STORAGE);  
    if (grant != PackageManager.PERMISSION_GRANTED) {  
        requestPermissions(new String[]{ Manifest.permission.READ_EXTERNAL_STORAGE }, 24);  
    } else {  
        // permission already granted  
        readFile();  
    }  
}
```

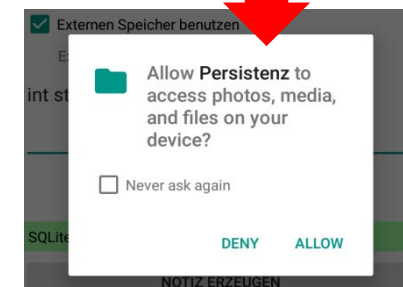
Request Code

Runtime-Check, ob benötigte
Permission(s) vorhanden,
sonst Permission(s) anfragen


Callback aus
Permission-Anfrage

```
@Override  
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {  
    switch (requestCode) {  
        case 24: // load file  
            if (grantResults.length > 0 && grantResults[0] != PackageManager.PERMISSION_GRANTED) {  
                Toast.makeText(this, "Permission " + permissions[0] + " denied!", Toast.LENGTH_SHORT).show();  
            } else {  
                // permission was granted  
                loadFile();  
            }  
            break;  
    }  
}
```

Library-Tipp: hotchemi's PermissionsDispatcher
<https://github.com/hotchemi/PermissionsDispatcher>



Java-Repetition: Streams, Reader & Co (PRG2)

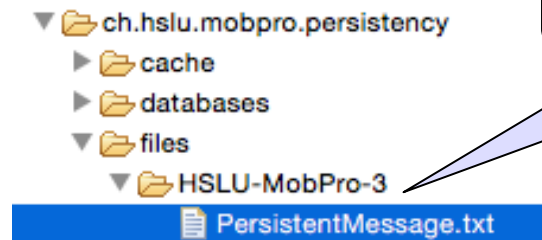
- Stream: Byte Datenstrom `[28 , 11 , 200 , 255 , 2 , 15 , 33]`
 - Auf File öffnen 
 - `FileOutputStream, FileInputStream`
- Stream kann in Zeichenstrom `['h' , 'a' , 'l' , 'l' , 'o']` umgewandelt werden
 - `FileReader, FileWriter` + „Buffered“-Versionen
- Immer schliessen!
 - `stream.close() , reader.close()`
- Nicht vergessen: `try-catch-finally` implementieren

Demo: Persistenz mit Datei

■ Text persistent speichern

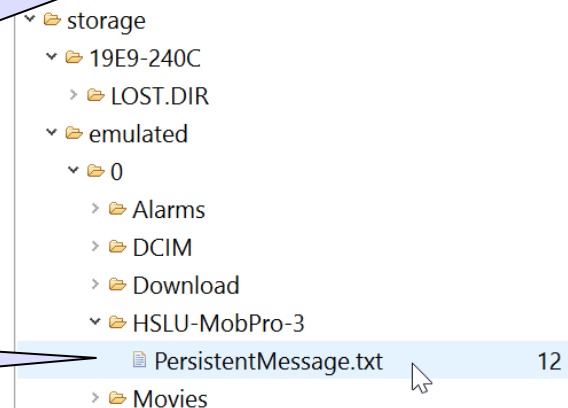
```
Writer writer = null;
try {
    writer = new BufferedWriter(new FileWriter(outFile));
    writer.write(text);
    return true;
} catch (final IOException ex) {
    // ...
} finally {
    Log.e("HSLU-MobPro-Persistenz", "Got a problem");
    // ...
}
```

■ Anschauen im „File Explorer“ („Android Device Monitor“, für Emulator)



Pfad im Applikationsverzeichnis (d.h. privater Speicher)

Pfad auf emulierter SD-Karte (d.h. «externer», öffentlicher Speicher)



Datei-System

Dieser Text wird von dieser Applikation persistent gespeichert, ich kann waehlen WO...

☐ Externen Speicher benutzen

SPEICHERN

LADEN

Hinweis: Für Android-Emulator muss SD-Card im Konfigurationsmenu (advanced) aktiviert sein

The image shows the Android Studio interface with the 'Virtual Device Configuration' dialog open. The 'Memory and Storage' tab is selected, and the 'SD card' option is set to 'Studio-managed' with a size of 100 MB. A red box highlights this setting. To the right, an Android emulator is shown with the 'Storage & USB' screen. The 'Internal storage' is 172 MB used of 1.94 GB. The 'Portable storage' section shows an 'SDCARD' that is 'Ejected'. A red box highlights the 'SDCARD' section. A red dotted arrow points from the 'SDCARD' section to a file explorer view of the emulator's storage. The file explorer shows a directory structure: 'storage' -> '19E9-240C' -> 'emulated' -> '0'. A speech bubble points to this path with the text 'Im Emulator nicht zugreifbar?'. Another speech bubble points to the 'storage' directory with the text 'Im Emulator kommt dieses Verzeichnis mit Environment.getExternalStorageDirectory() zurück'. The bottom right corner of the emulator shows a file named 'PersistentMessage.txt'.

Android Virtual Device Manager

Your Virtual Devices
Android Studio

Virtual Device Configuration

Android Virtual Device (AVD)
Verify Configuration

Emulated Performance

Use Host GPU ☒ Store a snapshot for faster startup ☐
You can either use Host GPU or Snapshots

Memory and Storage

RAM: 1536 MB
VM heap: 64 MB
Internal Storage: 200 MB
SD card: ☒ Studio-managed 100 MB
☐ External file

Custom skin definition
nexus_5
[How do I create a custom hardware skin?](#)

Keyboard ☒ Enable keyboard input

Hide Advanced Settings

+ Create Virtual Device...

Storage & USB

Device storage
172 MB
Total used of 1.94 GB

Internal storage
172 MB used of 1.94 GB

Portable storage

SDCARD
Ejected

Im Emulator nicht zugreifbar?

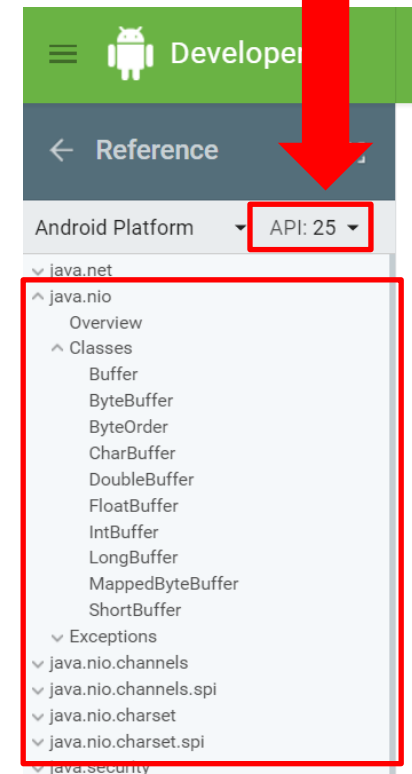
Im Emulator kommt dieses Verzeichnis mit `Environment.getExternalStorageDirectory()` zurück

storage
19E9-240C
LOST.DIR
emulated
0
Alarms
DCIM
Download
HSLU-MobPro-3
PersistentMessage.txt
Movies

...java.nio.file.Path?

- Mit Java 7 wurden die I/O Klassen im JDK grundlegend überarbeitet (`java.nio` = «new IO»)
 - Anstelle von `java.io.File` neu `java.nio.file.Path`
- Android unterstützt alle Language Features von Java 7, aber nur ein Subset des Java-API (Klassenbibliothek)
- D.h. `Path` ist für Android nicht verfügbar
 - Aber ab API 26 (Android 8) schon!
 - Nur leider noch nicht verbreitet genug...

Update: Verfügbar ab
Android 8 / API 26

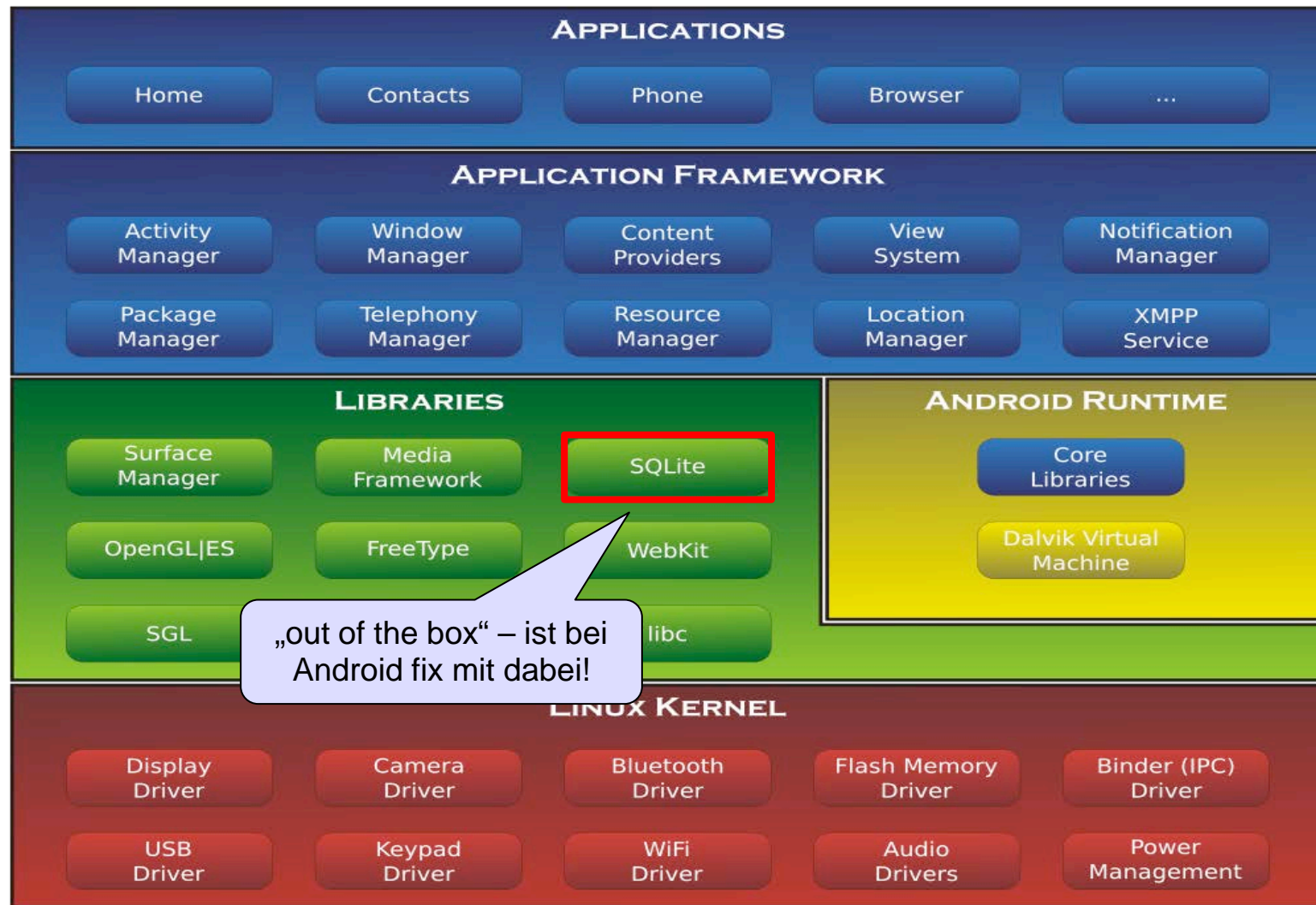




<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

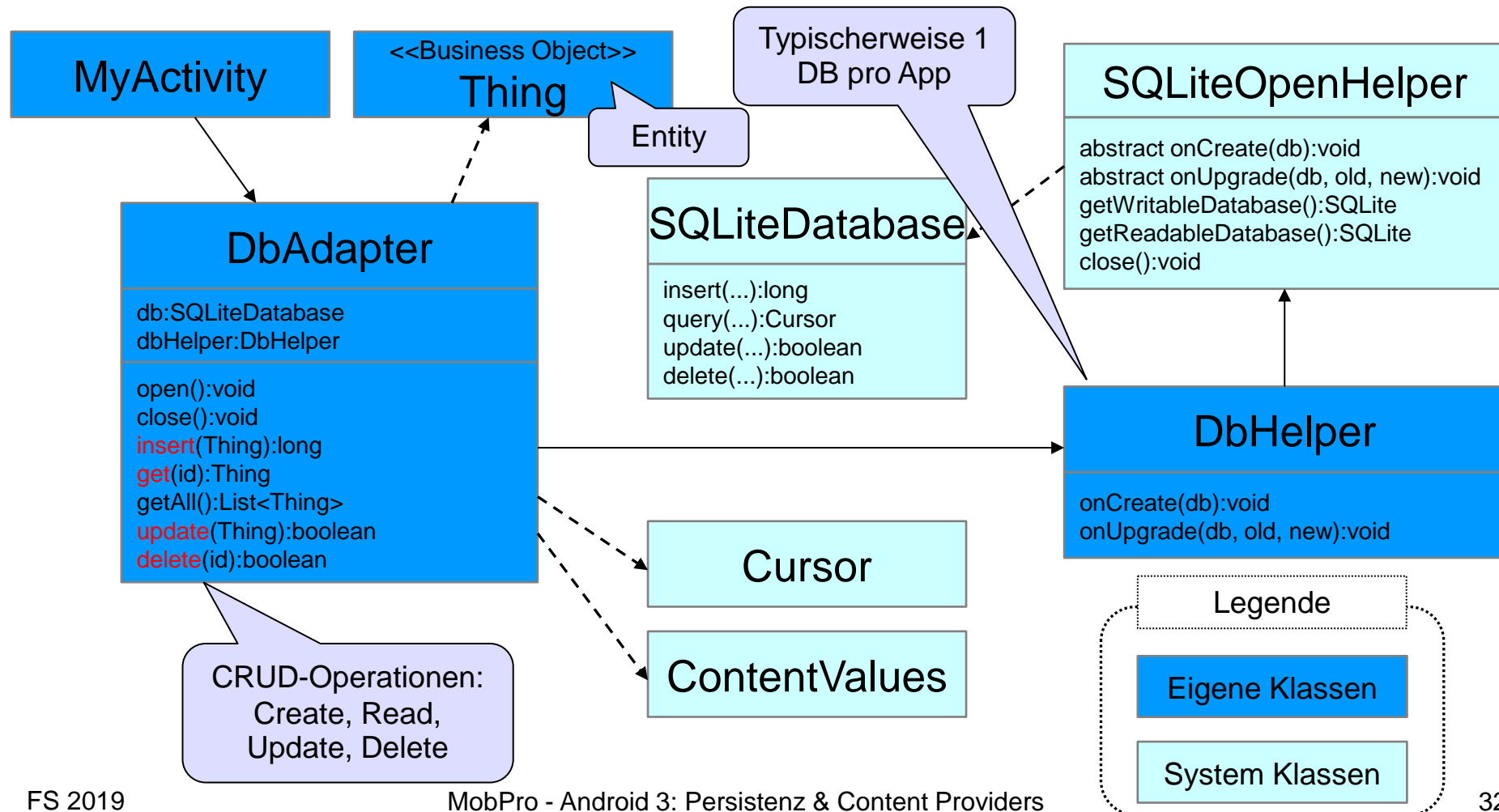
Persistenz: Datenbank (Room)

Die Android-DB: SQLite



SQLite Framework

- DbAdapter = Verbindung Business-Objekte <-> DB



Die harte Tour: SQLite in Rohform

Save data using SQLite

Saving data to a database is ideal for repeating or structured data, such as contact information. This page assumes that you are familiar with SQL databases in general and helps you get started with SQLite databases on Android. The APIs you'll need to use a database on Android are available in the `android.database.sqlite` package.

- ! **Caution:** Although these APIs are powerful, they are fairly low-level and require a great deal of boilerplate code. *Wer's gerne aufwändig und fehleranfällig mag...*
- There is no compile-time verification of raw SQL queries. As your data graph changes, you need to update the affected SQL queries manually. This process can be time consuming and error prone.
 - You need to use lots of boilerplate code to convert between SQL queries and data objects.

For these reasons, we **highly recommended** using the Room Persistence Library as an abstraction layer for accessing information in your app's SQLite databases.

<https://developer.android.com/training/data-storage/sqlite.html>

Room: Ein objektrelationaler Mapper

- Room ist ein ORM für Android
 - Klassen werden auf relationale DB-Tabellen gemappt
 - Zugriff auf Datenbank wird abstrahiert

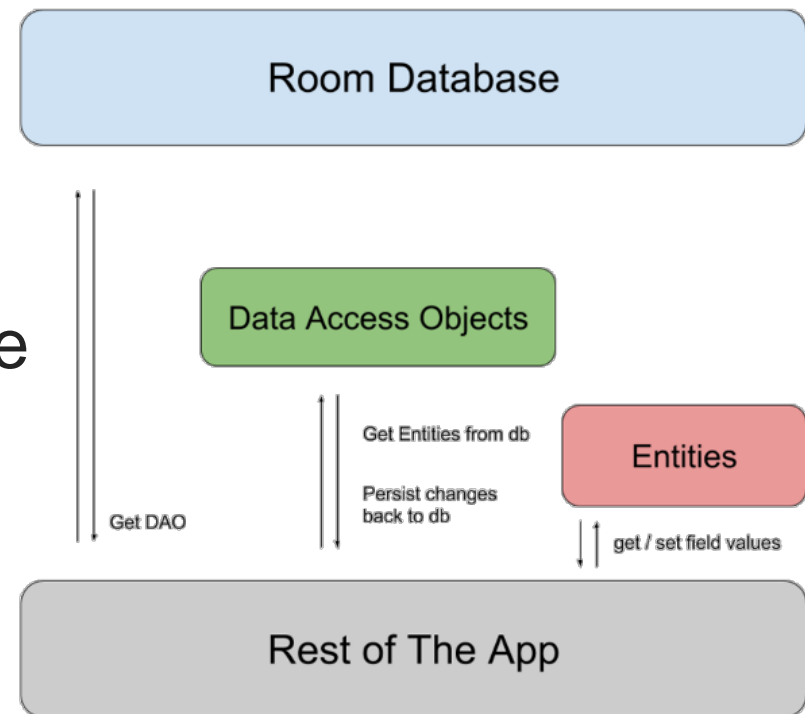
Typischerweise werden SQL-Statements durch Methodenaufrufe gekapselt

- Spezialfälle des *Room* ORM:
 - Datenzugriff über DAO: Queries werden als SQL-Statements in Annotationen definiert
 - Beziehungen zwischen Entitäten müssen manuell abgebildet werden (Performance!)
 - Nested Objects: Mehrere POJOs in einer Tabelle
 - Einschränkungen für Datenzugriffe

Standardmässig nicht möglich im UI Thread. Nebenläufigkeit folgt!

Die drei Room-Komponenten

- **Database**
Abstraktion der Datenbankverbindung
- **Entity**
Repräsentation einer Tabelle in der relationalen DB
- **DAO**
(Data Access Object)
Enthält Methoden für Datenzugriff



Room – Code-Beispiele

Entity: POJO mit
Annotationen

DAO: Datenzugriff über Annotationen
(teilweise mit SQL-Queries)

```
@Entity
public class User {
    @PrimaryKey
    public int uid;

    @ColumnInfo(name = "first_name")
    public String firstName;

    @ColumnInfo(name = "last_name")
    public String lastName;
}
```

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Insert
    void insertAll(User... users);

    @Delete
    void delete(User user);
}
```

Room – Code-Beispiele

Database: Subklasse von *RoomDatabase*,
konfiguriert mit *Database* Annotation

Version ist wichtig für
Migration!

```
@Database(entities = {User.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UserDao userDao();
}
```

Eine Instanz der DB erzeugen

```
AppDatabase db = Room.databaseBuilder(
    getApplicationContext(),
    AppDatabase.class,
    "database-name»
).build();
```

Daten mit Entitäten definieren

- POJO mit **@Entity** Annotation
- Primärschlüssel (wird in jeder Entität benötigt)
 - **@PrimaryKey** für einzelnes Feld...
 - ... optional mit **autoGenerate** Property
 - Für zusammengesetzte Primärschlüssel: **primaryKey** Property in **@Entity** Annotation
- Falls bestimmte Felder nicht gespeichert werden sollen
 - **@Ignore** Annotation für einzelnes Feld
 - Mit **ignoredColumns** Property in **@Entity** Annotation für mehrere Felder (v.a. von Superklassen)

Code-Beispiel

```
@Entity(primaryKeys = {"firstName", "lastName"},
        ignoredColumns = "password, otherField")
public class User extends Party {
    @PrimaryKey(autoGenerate = true)
    public int id;

    public String firstName;
    public String lastName;

    @Ignore
    Bitmap picture;
}
```

Achtung!

Dieses Code-Beispiel definiert mehrere Primärschlüssel und vermischt Ansätze zum Ignorieren von Feldern zwecks Syntax-Demonstration!

Beziehungen modellieren

Define relationships between objects

Performanzgründe

Because SQLite is a relational database, you can specify relationships between objects. Even though most object-relational mapping libraries allow entity objects to reference each other, Room explicitly forbids this. To learn about the technical reasoning behind this decision, see [Understand why Room doesn't allow object references](#).

```
@Entity(foreignKeys = @ForeignKey(entity = User.class,  
    parentColumns = "id",  
    childColumns = "user_id"))  
  
public class Book {  
    @PrimaryKey  
    public int bookId;  
  
    public String title;  
  
    @ColumnInfo(name = "user_id")  
    public int userId;  
}
```

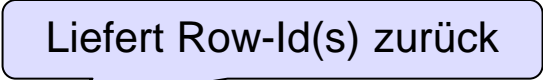

Feld-Typ: Nur ID, nicht User

Mit DAOs auf Daten zugreifen

- Data Access Objects (DAOs) enthalten Methoden für den abstrahierten Datenbankzugriff
- Dies trägt zur *Separation of Concerns* bei und erhöht die Testbarkeit (DAOs können gemockt werden)
- DAOs werden als Interfaces oder abstrakte Klassen definiert → Room erzeugt passende Implementationen bei der Kompilierung!
- Zwei Möglichkeiten:
 - Convenience queries
 - **@Query** Annotation mit SQL-Statements

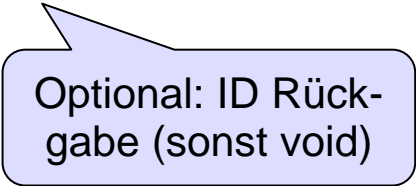
Typischerweise eine DAO-Klasse mit pro Entity, mit allen möglichen Operationen

Convenience Queries

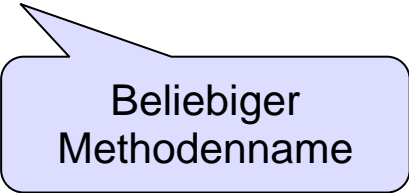
- Werden über Annotations für die jeweiligen Methoden definiert: **@Insert**, **@Update**, **@Delete**
- Alle Parameter müssen Klassen mit einer **@Entity** Annotation (oder Collections/Arrays) davon sein
- Rückgabewerte
 - Insert: **long** bzw. **long[]** bzw. **List<Long>**

 - Update / Delete: **int**


@Insert

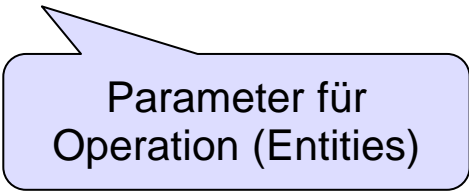
```
public long[] insertUsersAndFriends(User user, List<User> friends);
```



Optional: ID Rückgabe (sonst void)



Beliebiger
Methodenname



Parameter für
Operation (Entities)

Convenience Queries: Weitere Beispiele

```
@Dao
public interface MyDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    public void insertUsers(User... users);

    @Insert
    public void insertBothUsers(User user1, User user2);

    @Insert
    public long[] insertUsersAndFriends(User user, List<User> friends);

    @Update
    public void updateUser(User... users);

    @Delete
    public void deleteUser(User... users);
}
```

Custom Queries mit @Query

- Die @Query Annotation kann für Schreib- und Lesevorgänge genutzt werden
- Jede @Query wird zur Kompilierzeit überprüft
→ Kompilierfehler bei ungültigen Queries
- Für eine @Query kann eine beliebige Anzahl (0..n) Parameter verwendet werden
- Wenn nicht ganze Objekte benötigt werden, können durch die Verwendung von POJOs mit @ColumnInfo Annotationen Ressourcen gespart werden

Keine Laufzeitfehler!

Custom Queries: Codebeispiele

```
@Dao
public interface MyDao {
    @Query("SELECT * FROM user")
    public User[] loadAllUsers();

    @Query("SELECT * FROM user WHERE age > :minAge")
    public User[] loadAllUsersOlderThan(int minAge);

    @Query("SELECT first_name, last_name FROM user
            WHERE region IN (:regions)")
    public List<NameTuple> loadUsersFromRegions(List<String> regions);
}

public class NameTuple {
    @ColumnInfo(name = "first_name")
    public String firstName;

    @ColumnInfo(name = "last_name")
    public String lastName;
}
```

DB-Einträge in einer Liste darstellen

- Verschiedene Möglichkeiten, je nach Umfang / Komplexität der Datensätze:
 - ListView (siehe nächste Folie)
 - RecyclerView
<https://developer.android.com/guide/topics/ui/layout/recyclerview>
 - Auch in Kombination mit ViewModel und LiveData
<https://codelabs.developers.google.com/codelabs/android-room-with-a-view>
- In jedem Fall werden spezifische Adapter benötigt, um die Daten auf Views zu mappen

DB-Einträge in einer Liste darstellen

```
public class UsersAdapter extends ArrayAdapter<User> {  
    public UsersAdapter(Context ctx, User[] users) { super(ctx, 0, users); }
```

@Override

```
public View getView(int position, View view, ViewGroup parent) {
```

1 User userItem = getItem(position);

2 if (view == null) {

view = LayoutInflater.from(getContext())
.inflate(R.layout.userview_layout, parent);

}

3 // TODO: populate fields/sub-views of view with data of userItem

return view;

}

}

TextView name = view.findViewById(R.id.name);
name.setText(user.getName());

```
public class UsersListActivity extends ListActivity {
```

@Override

```
protected void onCreate(final Bundle savedInstanceState) {
```

super.onCreate(savedInstanceState);

final Users[] users = userDao().getAllUsers();

final UsersAdapter adapter = new UsersAdapter(this, users);

setListAdapter(adapter);

}

}

Room: Weitere Themen

- Weitere Themen, die für Android Apps mit Room relevant sein könnten:
 - Queries in Klassen kapseln (Views)
<https://developer.android.com/training/data-storage/room/creating-views>
 - Observable Queries mit Live Data
<https://developer.android.com/training/data-storage/room/accessing-data#query-observable>
 - Datenbank migrieren (z.B. bei App Updates)
<https://developer.android.com/training/data-storage/room/migrating-db-versions>
 - Datenbank testen
<https://developer.android.com/training/data-storage/room/testing-db>
 - TypeConverter: Objekt-Referenzen in der Datenbank
<https://developer.android.com/training/data-storage/room/referencing-data>

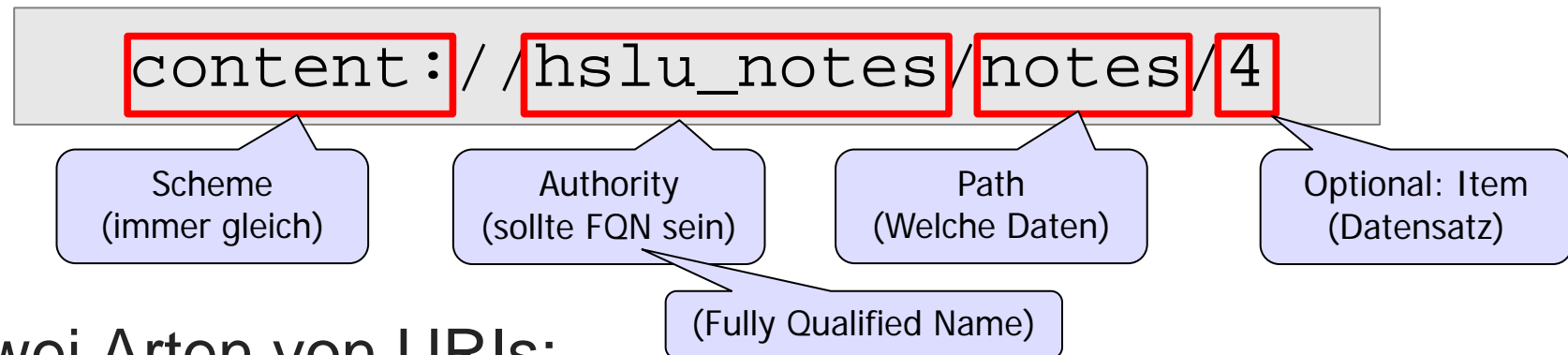


<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Content Providers

Content Provider

- Content Provider stellen für andere Applikationen Daten bereit
 - Daten stammen aus einer gekapselten DB oder aus dem privaten Dateisystem oder werden on-the-fly erzeugt
 - Zugriff auf Daten über URI (Uniform Resource ID), z.B.:



- Zwei Arten von URIs:
 - Pfad (Bezeichnet Datenmenge, vgl. Verzeichnis mit Dateien)
 - Item (Einzelnes Datenelement, vgl. einzelne Datei)

Standard Content Providers

- Im Android-System gibt es bereits einige Content Providers, die benutzt werden können:
 - Kontakte: Namen, Telefon-Nummern, Emails, Adressen, etc.
 - SMS/MMS: Erhaltene/Gesendete/Draft SMS/MMS
 - Media Store: Auf Device gespeicherte Audio-, Video-, Bilder-Daten
 - Settings: Einstellungen für das Gerät
 - Kalender: Kalender, Events, Erinnerungen, Teilnehmer, etc.
- Daten sind meist in mehreren Tabellen abgelegt

Exkurs: REST-ful Webservices

Content-Provider API
lehnt sich stark
an dieses Modell an

- Webservice auf der Basis von HTTP
- Grundidee (in purer Form)
 - URL einer Ressourcensammlung (<http://directory.com/contacts/>)
oder URL einer einzelnen Resource (<http://directory.com/contacts/17>)
 - HTTP Methode = Operation auf Daten (GET, PUT, POST, DELETE)
 - Antwort-Datenformat = XML, JSON, ...

Resource	GET	PUT	POST	DELETE
Collection URI, such http://directory.com/contacts/	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Create a new entry in the collection. The new entry's URL is usually returned by the operation.	Delete the entire collection.
Element URI, such as http://directory.com/contacts/17	Retrieve a representation of the addressed collection member, expressed in an appropriate media type.	Replace the addressed member of the collection, or if it doesn't exist, create it.	Treat the addressed member as a collection in its own right and create a new entry in it.	Delete the addressed member of the collection.

Quelle: http://en.wikipedia.org/wiki/Representational_state_transfer#Example

Content Resolver & Content Provider

- Zugriff auf einen Content Provider erfolgt über einen Content Resolver
 - `Context.getContentResolver()`
 - Bietet DB-Methoden und Zugriff auf Content via Streams
 - CRUD: `insert()` / `query()` / `update()` / `delete()`
 - `openInputStream(uri)` / `openOutputStream(uri)`
 - Ein Content Resolver ist ein Proxy, der...
 - URI auflöst und zuständigen Content Provider sucht/findet
 - Interprozess-Kommunikation behandelt (aufrufende App ist meist in einem anderen Package als der aufgerufene CP)
- Achtung: Permissions müssen u.U. gesetzt werden!
`<uses-permission android:name="android.permission.READ_CALENDAR" />`

Zugriff auf Daten

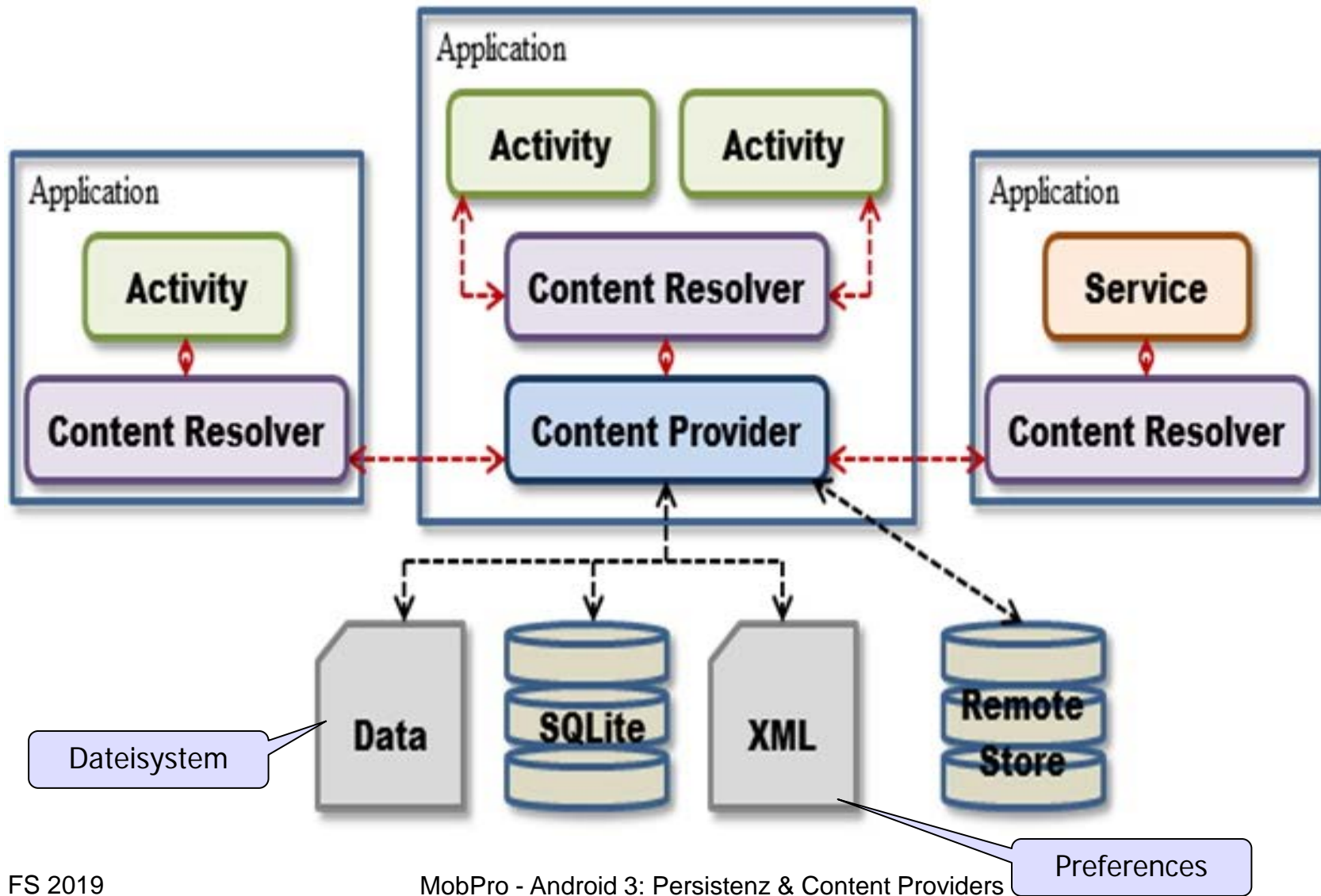
■ Über Content Resolver + Query:

```
Cursor cursor = getContentResolver().query(  
    contentUri,          // The content URI of the table  
    projection,          // The columns to return for each row  
    selectionClause,     // Selection criteria  
    selectionArgs,       // Selection criteria  
    sortOrder);         // The sort order for the returned row
```

Vergleich ContentProvider Query und SQL Query Parameter:

Content Provider Query	SQL SELECT Query	Notes
contentUri	FROM table_name	contentUri maps to the table in the provider named table_name.
projection	Col, col, col,...	projection is an array of columns that should be included for each row retrieved.
selection	WHERE col = value	selection specifies the criteria for selecting rows.
selectionArgs	(No exact equivalent. Selection arguments replace ? placeholders in the selection clause.)	-
sortOrder	ORDER BY col,col,...	sortOrder specifies the order in which rows appear in the returned Cursor.

Content Provider: Anwendung

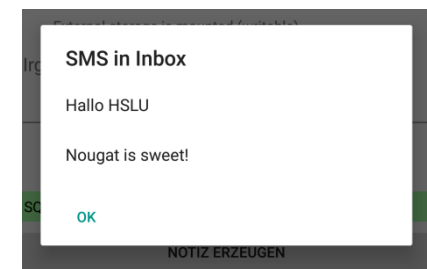


Beispiel: SMS Provider

Benötigt Permission: Manifest.permission.READ_SMS!
Vor der Ausführung des Codes testen und ggf. beantragen!

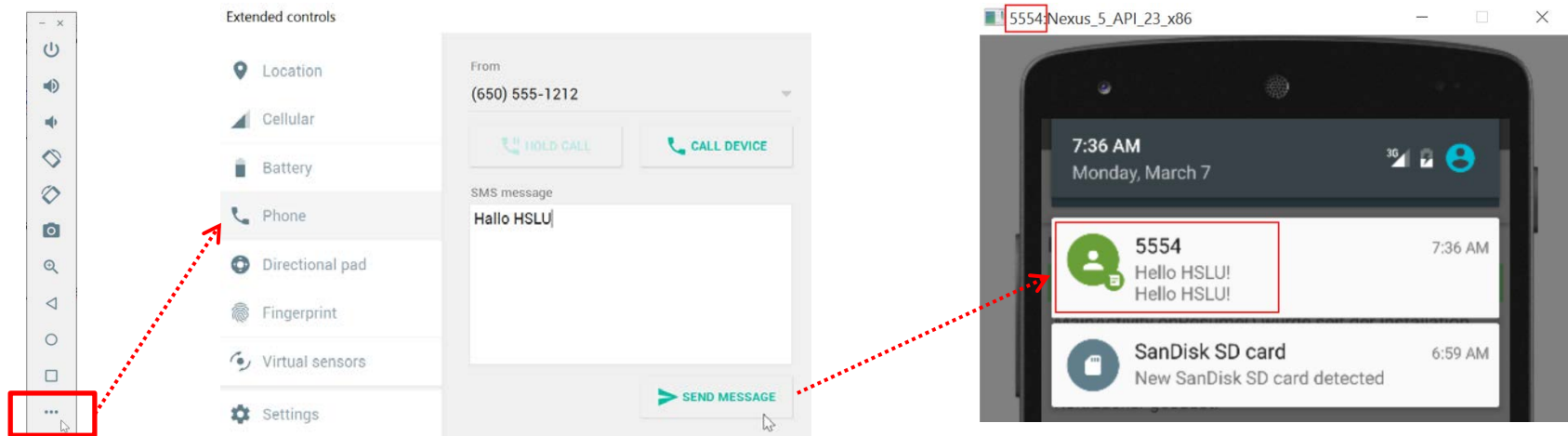
- SMS des Systems sind über Content-Provider zugänglich
 - `android.provider.Telephony.Sms`
 - «Sub Providers» für *Sent*, *Inbox*, *Draft*, etc.
 - Im Package `android.provider.*` finden wir „Contract Klasse“ `Telephony.Sms` mit Hilfsklassen `BaseColumns` und `Telephony.TextBasedSmsColumns`
 - Hier finden wir Content-URI und Spalten-Namen für Projections
- Anwendungsbeispiel : Alle Sms mit Text anzeigen

```
public void showSmsList(final View view) {  
    final Cursor cursor = getContentResolver().query(  
        Telephony.Sms.Inbox.CONTENT_URI, // content uri  
        new String[]{ Telephony.Sms.Inbox._ID, Telephony.Sms.Inbox.BODY }, // projection  
        null, null, null); // selection, selection args, sort order  
  
    new AlertDialog.Builder(this)  
        .setTitle("SMS in Inbox")  
        .setCursor(cursor, null, Telephony.TextBasedSmsColumns.BODY)  
        .setNeutralButton("Ok", null)  
        .create()  
        .show();  
}
```



SMS an Emulator schicken

- Bei Ausführung des Beispielcodes sehen wir im Emulator nichts. Grund: keine SMS vorhanden
- Mit den *Extended Controls* können SMS an den Emulator geschickt werden. Nummer angeben!



Tipp: Wird als Absender eine Emulatornummer verwendet, so können SMS zwischen zwei Emulatoren verschickt werden (Reply-Funktion)

Einen Content Provider verwenden

API? Datenstruktur?

- Wo fange ich an?
 - Jeder Content Provider hat zwar ein Standard-API, aber woher weiss ich die Content-URI, Projection, etc.?
- Dokumentation?
 - In Android Doku sind Zugriff auf Kontakte und Kalender gut dokumentiert (weil eher kompliziertes Modell)
 - <http://developer.android.com/guide/topics/providers/calendar-provider.html>
 - <http://developer.android.com/guide/topics/providers/contacts-provider.html>
 - Einstiegspunkt = `Package android.provider.*`
 - <http://developer.android.com/reference/android/provider/package-summary.html>
 - Ausgangspunkt: «Contract Klassen» mit Content-URI und Column-Constants

Tutorials

Für andere Provider

Eigener Content Provider

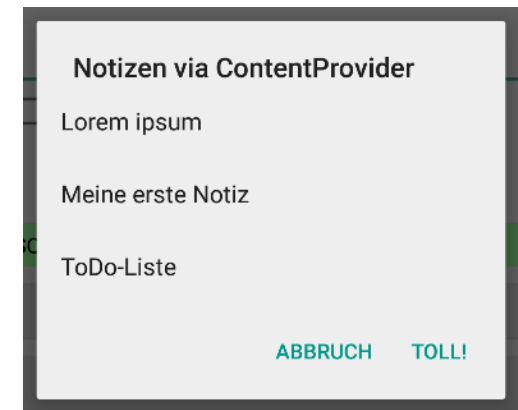
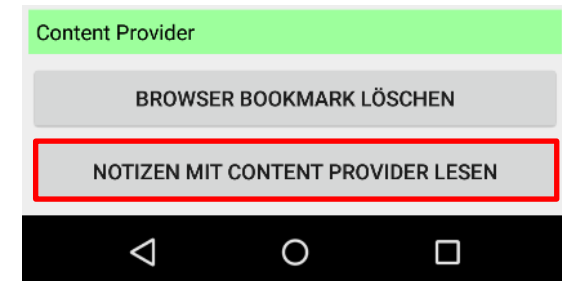
- Einen eigenen Content Provider zu schreiben ist nicht so schwer
- Die eigene Klasse muss von der abstrakten Klasse `android.content.ContentProvider` ableiten
- Wird bei Start der App hochgefahren und bleibt aktiv
 - In `onCreate()` kann eine Initialisierung vorgenommen werden (einzige Lifecycle-Methode)
- CRUD-Methoden: `query`, `insert`, `update`, `delete`
 - Nicht alle müssen implementiert werden

So kann z.B. ein read-only Content Provider angelegt werden

Demo: Content Provider für Notizen

- Dialog zeigt Notizen an
- Nur für internen Gebrauch
 - `exported=false`
- NotesProvider: Konstanten definiert in NotesContract
- Aufrufender Code in Activity:

```
public void readNotesFromContentProviderOnClick(final View view) {  
    final Cursor cursor = getContentResolver().query(  
        Uri.parse(NotesContract.CONTENT_URI), NotesContract.PROJECTION,  
        null, // SELECT *  
        null, NotesContract.COLUMN_TITLE);  
  
    new AlertDialog.Builder(this).setTitle("Notizen via ContentProvider")  
        .setCursor(cursor, null, NotesContract.COLUMN_TITLE)  
        .setPositiveButton("Toll!", null)  
        .setNegativeButton("Abbruch", null).create().show();  
}
```





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Übung 3

Zur Übung 3

- Preferences
 - Resume Counter
 - Tee Präferenzen inkl. Default
- Dateisystem
 - Text speichern / laden
 - intern / extern
- SQLite DB / Room
 - Notizen erfassen & anzeigen
 - Inkl. DBAdapter & DbHelper
- Content Provider
 - SMS Anzeigen

Optionaler Teil

D.h. müssen Sie nicht
vorzeigen für's Testat.
- Ist aber natürlich
trotzdem Prüfungstoff!

