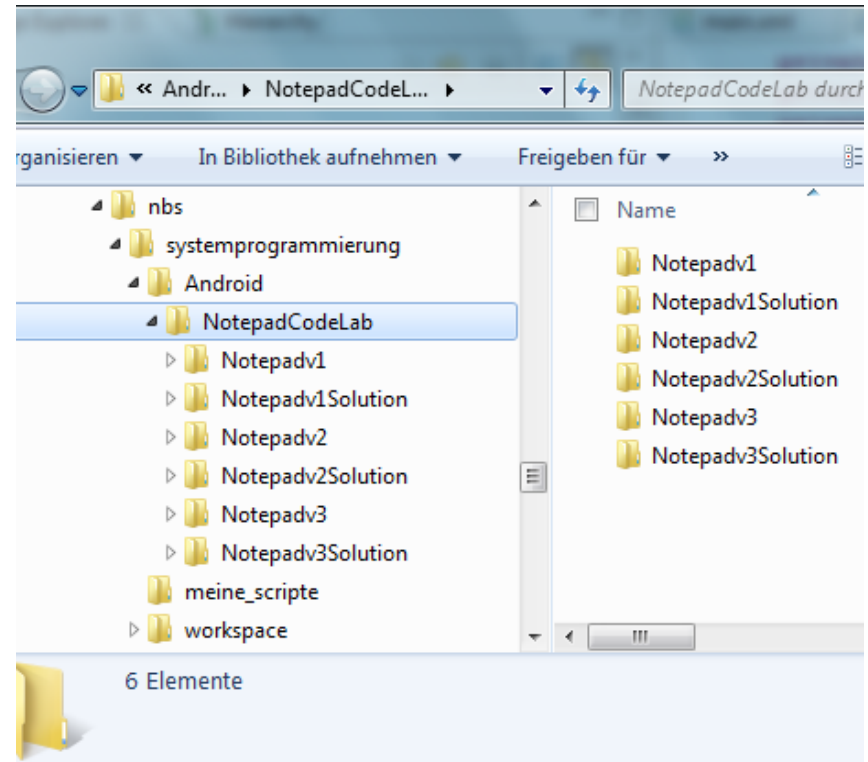


Systemprogrammierung Notepad Tutorial Handson

Prof. Dr. Ing. Birgit Wendholt
HAW-Hamburg Department Informatik

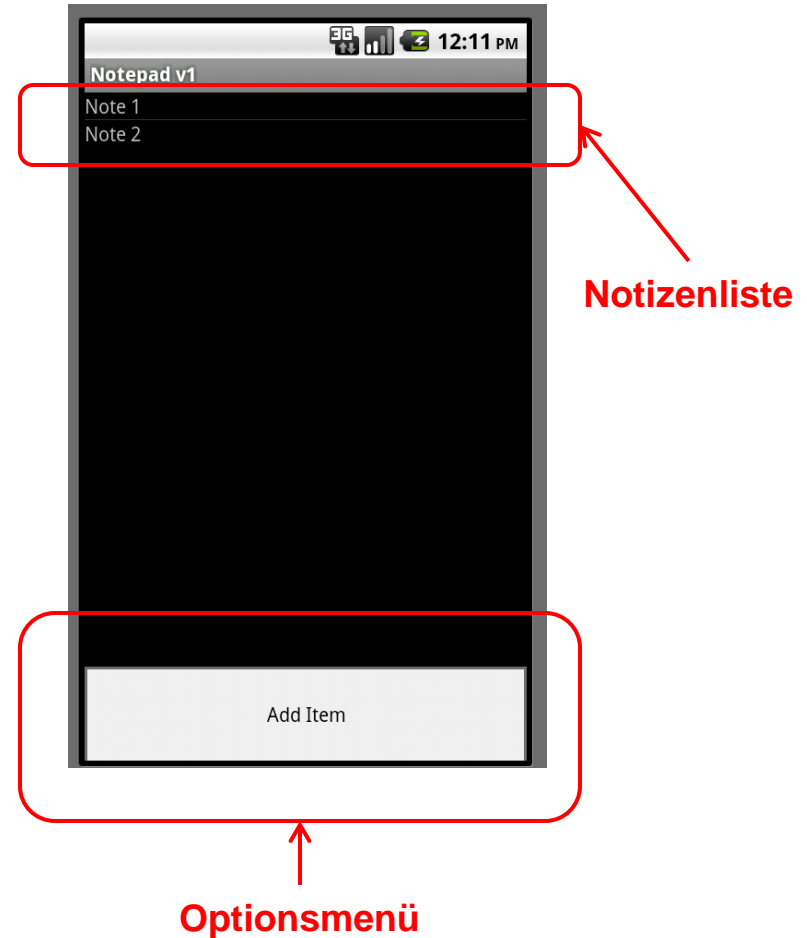
- **Ziel:** Am Beispiel das Einrichten eines Android Projektes und das Entwickeln einer Applikation mit einer nicht trivialen Benutzerschnittstelle kennenlernen.
- **Ergebnis:** Kenntnis der wichtigsten Konzepte in der Entwicklung von Android Applikationen.
- Komplexere Implementierung im Abschnitt [Sample Code](#).
- **Aufbau:**
 - **Übung 1:** Eine Notizliste erzeugen und neue Notizen erzeugen. ListActivity sowie Erzeugen und Umgang mit Optionsmenüs kennenlernen. Eine SQLite Datenbank zum Speichern von Notizen verwenden.
 - **Übung 2:** 2'te Aktivität in einer Applikation verwenden. Daten zwischen Aktivitäten austauschen und komplexere Layouts verwenden.
 - **Übung 3:** Die Anwendung von Lifecycle Methoden, um den Zustand einer Applikation während des gesamten Lifecycles zu erhalten.

- Download der [Archivs mit den Übungen \(.zip\)](#) von der Android Seite.
- Entpacken des Archivs.
- **NotepadCodeLab** Ordner öffnen:
 - Notepadv1, Notepadv2, Notepadv3: vorbereitete Android Projekte, die vervollständigt werden müssen.
 - Notepadv1Solution, Notepadv2Solution, Notepadv3Solution: Lösungen, wenn nichts mehr geht 😊.



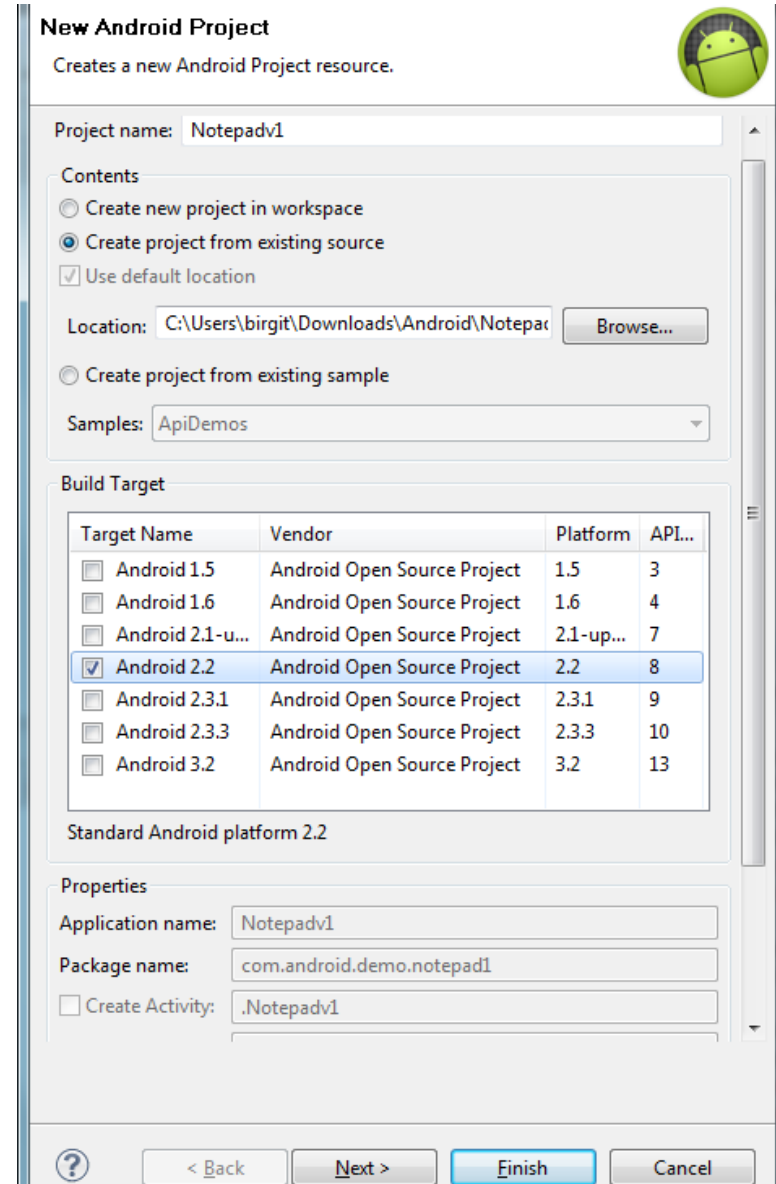
- **Lernziele:**

- Umgang mit ListActivities
- Arbeiten mit Optionsmenüs
- Nutzen einer SQLite Datenbank, um Notizen zu speichern
- Binden von Daten einer Datenbank an eine *ListView* → *SimpleCursorAdapter*
- Bildschirmlayout, insbesondere das Layout einer *ListView*
- Hinzufügen von Menüitems in ein Optionsmenü
- Behandeln von Menüselektionen



Übung 1: Schritt 1

- Neues Android Projekt anlegen: **File > New > Android Project.**
- **Create project from existing source.**
- **Browse:** Notepadv1 im Ordner NotepadCodeLab auswählen.
- Build Target auswählen: Android 2.2, Min SDK 8
- **Finish**
- **Fehler: Android Tools > Fix Project Properties.**



New Android Project
Creates a new Android Project resource.

Project name:

Contents

☐ Create new project in workspace

☒ Create project from existing source

☒ Use default location

Location:

☐ Create project from existing sample

Samples:

Build Target

| Target Name | Vendor | Platform | API... |
|-------------------------------------------------|-----------------------------|-----------|--------|
| <input type="checkbox"/> Android 1.5 | Android Open Source Project | 1.5 | 3 |
| <input type="checkbox"/> Android 1.6 | Android Open Source Project | 1.6 | 4 |
| <input type="checkbox"/> Android 2.1-u... | Android Open Source Project | 2.1-up... | 7 |
| <input checked="" type="checkbox"/> Android 2.2 | Android Open Source Project | 2.2 | 8 |
| <input type="checkbox"/> Android 2.3.1 | Android Open Source Project | 2.3.1 | 9 |
| <input type="checkbox"/> Android 2.3.3 | Android Open Source Project | 2.3.3 | 10 |
| <input type="checkbox"/> Android 3.2 | Android Open Source Project | 3.2 | 13 |

Standard Android platform 2.2

Properties

Application name:

Package name:

☐ Create Activity:

Übung 1 – Schritt 2

- **NotesDbAdapter** Klasse:

- kapselt den Datenzugriff für eine SQLite Datenbank für Notizen.
- **Konstanten:** Namen in der Datenbank (Tabellen, Spalten), ein String für das Erzeugen eines Datenbankschemas.
- Datenbankname “data” mit der Tabelle “notes” und den Spalten _id, title, body.
- Konstruktor erhält einen Kontext → Zugriff zum Android Betriebssystem.

- **Methoden:**

- **open()**
- **close()**
- **createNote()**
- **deleteNote()**
- **fetchAllNotes()**
- **fetchNote()**
- **updateNote()**

```
public class NotesDbAdapter {  
    public static final String KEY_TITLE = "title";  
    public static final String KEY_BODY = "body";  
    public static final String KEY_ROWID = "_id";  
    private static final String TAG = "NotesDbAdapter";  
    private DatabaseHelper mDbHelper;  
    private SQLiteDatabase mDb;  
  
    private static final String DATABASE_CREATE =  
        "create table notes (_id integer primary key autoincrement, "  
        + "title text not null, body text not null);";  
  
    private static final String DATABASE_NAME = "data";  
    private static final String DATABASE_TABLE = "notes";  
    private static final int DATABASE_VERSION = 2;  
  
    private final Context mContext;  
    private static class DatabaseHelper extends SQLiteOpenHelper {  
  
        DatabaseHelper(Context context) {  
            super(context, DATABASE_NAME, null, DATABASE_VERSION);  
        }  
    }  
    ... }
```

Übung 1 – Schritt 2

- **open()**: ruft die Methode **getWritableDatabase()** des **DatabaseHelper** für das Erzeugen und Öffnen der DB übernimmt. Lokale Implementierung der abstrakten Klasse **SQLiteOpenHelper**.
- **close()**: schließt die Datenbank

```
public NotesDbAdapter open() throws  
    SQLException {  
    mDbHelper = new  
        DatabaseHelper(mCtx) ;  
    mDb = mDbHelper.  
        getWritableDatabase() ;  
    return this ;  
}
```

```
public void close() {  
    mDbHelper.close() ;  
}
```


Übung 1- Schritt 2

- ***createNote()***: erzeugt eine neue Note in der Datenbank. Gibt die `_id` der erzeugten Notiz zurück.
- ***deleteNote()***: löscht eine Notiz über deren id.

```
public long createNote(String title,
    String body) {
    ContentValues initialValues =
        new ContentValues();
    initialValues.put(KEY_TITLE, title);
    initialValues.put(KEY_BODY, body);

    return mDb.insert(DATABASE_TABLE,
        null, initialValues);
}

public boolean deleteNote(long rowId) {

    return mDb.delete(DATABASE_TABLE,
        KEY_ROWID + "=" + rowId,
        null) > 0;
}
```

- ***fetchAllNotes()*** erzeugt eine Query, um alle Notizen der DB über einen Cursor zu lesen.
- ***query()*** Parameter:
 - Name der DB Tabelle
 - Liste von Spalten, die gelesen werden sollen.
 - Filter (SQL WHERE)
 - Filter-Argumente
 - Gruppierung (SQL GROUP BY)
 - Having (SQL HAVING)
 - Anordnung (SQL ORDER BY)
 - Für ***null*** Parameter werden Defaults verwendet → SQLiteDatabase

```
public Cursor fetchAllNotes() {  
    return mDb.query(DATABASE_TABLE,  
        new String[] {  
            KEY_ROWID,  
            KEY_TITLE,  
            KEY_BODY},  
  
        null,  
        null,  
        null,  
        null,  
        null);  
}
```

- **fetchNote()**: liefert einen Cursor auf die Notiz mit id *rowId*.
- **query()**:
 - Der erste Parameter zeigt, dass nur genau ein Ergebnis erwartet wird.
 - Der Filter Parameter selektiert das Ergebnis über die *id* einer Notiz.

```
public Cursor fetchNote(long rowId)
    throws SQLException {

    Cursor mCursor =
        mDb.query(true,
            DATABASE_TABLE,
            new String[]
                {KEY_ROWID,
                 KEY_TITLE,
                 KEY_BODY},
            KEY_ROWID + "=" + rowId,
            null,null,null,null, null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}
```

Übung 1 – Schritt 2

- **updateNote()**
 - wird mit **rowId**, **title** und **body** aufgerufen
 - verwendet ein ContentValues Objekt, um eine Notiz zu aktualisieren.

```
public boolean updateNote(long  
    rowId, String title, String body)  
{  
    ContentValues args =  
        new ContentValues();  
    args.put(KEY_TITLE, title);  
    args.put(KEY_BODY, body);  
  
    return mDb.update(  
        DATABASE_TABLE,  
        args,  
        KEY_ROWID + "=" + rowId,  
        null) > 0;  
}
```

Übung 1 – Schritt 3 + 4 - Layout erstellen

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

</LinearLayout>
```



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

    <ListView android:id="@android:id/list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView android:id="@android:id/empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/no_notes"/>

</LinearLayout>
```

Übung 1 – Schritt 3 + 4 - Layout erstellen

- Das @ Zeichen in den **id** Strings der **ListView** und **TextView** ist eine Instruktion für den XML Parser, den Rest des String zu expandieren und diesen als ID Ressource zu verwenden.
- **ListView** und **TextView** sind zwei alternative Views, von denen nur eine zu einer Zeit angezeigt wird:
 - **ListView**, wenn Notizen vorhanden sind.
 - **TextView** wenn keine Notizen vorhanden sind.
- Die ids **list** und **empty** sind im Android Package definiert, daher wird hier der Präfix **android:** verwendet (z.B. **@android:id/list**).
- Eine View mit einer **empty** Id wird automatisch angezeigt, wenn der **ListAdapter** einer **ListView** keine Daten hat.
- Die **android.R** Klasse definiert eine Menge von Ressourcen für alle Applikationen, die **R** Klasse eines Projektes die Ressourcen einer Applikation.

Übung 1 – Schritt 5 - Zeilen Layout einer Liste

- Für die Zeilen einer **ListView** muss ebenfalls eine Layout erzeugt werden.
- Erzeuge die Ressourcdatei **notes_row.xml** im Verzeichnis **res/layout** mit dem Inhalt rechts.
- Jede Notiz erhält in der **ListView** eine **TextView** für die Anzeige.
- Hier wird für die **TextView** eine Projekt-lokale ID **text1** verwendet. Das + nach dem @ in dem ID String bewirkt, dass die ID automatisch erzeugt wird.
- Im der Klasse **R.java** des Projektes finden sich nach dem Speichern Einträge für **notes_row** and **text1**.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView android:id="@+id/text1"

    xmlns:android="http://schemas.android.
        com/apk/res/android",

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"/>

public final class R {
    public static final class id {
        public static final int
            text1=0x7f050000;
    }
    public static final class layout {
        public static final int
            notepad_list=0x7f030000;
        public static final int
            notes_row=0x7f030001;
    }
}
```

Übung 1 – Schritt 6 – Notepadv1 Klasse

- Die Klasse **Notepadv1**, aktuelle eine ein **Activity** wird in eine **ListActivity** mit einem **ListAdapter** für das Verwalten der Notizen umgeschrieben.
- In der vorliegenden Version enthält die Klasse:
 - die Variable **mNoteNumber** zum Numerieren der Notizen.
 - Die überschriebenen Methoden **onCreate**, **onCreateOptionsMenu** und **onOptionsItemSelected**

```
public class Notepadv1 extends Activity {
    private int mNoteNumber = 1;

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

    @Override
    public boolean onCreateOptionsMenu(
        Menu menu) {
        return super.
            onCreateOptionsMenu(menu);
    }

    @Override
    public boolean
        onOptionsItemSelected(MenuItem
            item) {
        return super.
            onOptionsItemSelected(item);
    }
}
```


Übung 1 – Schritt 7+8 – ListActivity – onCreate

- **Notepadv1** leitet von **ListActivity** ab.
- **onCreate()** wird aufgerufen, wenn die Aktivität gestartet wird → Ressourcen der Aktivität initialisieren:
 - Initialisierung entlang der Vererbungshierarchie
 - Layout **notepad_list** zuordnen
 - **NotesDbAdapater** initialisieren und Öffnen der DB.
 - Füllen der Notizenliste mit **fillData**

```
public class Notepadv1 extends
    ListActivity {
    private int mNoteNumber = 1;
    private NotesDbAdapter mDbHelper;

    @Override
    public void onCreate(Bundle
        savedInstanceState) {
        super.onCreate(
            savedInstanceState);
        setContentView(
            R.layout.notepad_list);
        mDbHelper = new
            NotesDbAdapter(this);
        mDbHelper.open();
        fillData();
    }

    private void fillData() {

    }
}
```

Übung 1 – Schritt 9 - Optionsmenü

- **onCreateOptionsMenu** wird aufgerufen, wenn das Menü erzeugt wird.
- **onCreateOptionsMenu** legt die Struktur des Menüs fest. Dieses soll ein Item mit dem Text "Add Item" enthalten:
 - **strings.xml** Ressource unter **res/values** um einen String **"menu_insert"** erweitern
 - Konstante für die Menüitem Position definieren
 - Menüitem mit **add** hinzufügen; Parameter von **add**: ID einer Menügruppe (0-für keine), eindeutige ID für das Item, Ordnung der Items (0-keine Ordnung), String-Ressource für das Item.

```
<string name="menu_insert">
    Add Item</string>
```

```
public class Notepadv1 extends
    ListActivity
{
    public static final int INSERT_ID =
        Menu.FIRST;

    @Override
    public boolean onCreateOptionsMenu(
        Menu menu) {

        boolean result = super.
            onCreateOptionsMenu(menu);
        menu.add(0,
            INSERT_ID,
            0,
            R.string.menu_insert);

        return result;
    }
}
```

Übung 1 – Schritt 10

- ***onOptionsItemSelected*** reagiert auf Selektion-Events in einem Menü.
- Wird “Add Item” selektiert ruft Android die ***onOptionsItemSelected*** Methode mit dem selektierten item auf.
 - Wenn ***item.getId() == INSERT_ID***, soll eine neue Notiz erzeugt werden → Aufruf von ***createNote*** und Rückgabe von ***true***. (gelöst als switch - case)
 - Wenn die Item ID nicht bekannt ist, dann Delegation an ***super***.

```
@Override
public boolean
onOptionsItemSelected(
    MenuItem item) {
    switch (item.getItemId()) {
        case INSERT_ID:
            createNote();
            return true;
    }
    return super.
        onOptionsItemSelected(
            item);
}
```

Übung 1 – Schritt 11

- **createNote** Methode:
 - eine Notiz mit einem Namen und leerem Body erzeugen. Notizen werden automatisch durchnummeriert.
 - Notiz mit **mDbHelper.createNote** in die DB eintragen
 - **fillData** aufrufen, um die Notizenliste zu aktualisieren.

```
private void createNote() {  
    String noteName = "Note " +  
        mNoteNumber++;  
  
    mDbHelper.createNote(  
        noteName, "");  
  
    fillData();  
}
```

- **fillData** Methode:

- Verwenden eines **SimpleCursorAdapter** um den DB Cursor an die Zeilen der **ListView** zu binden, in diesem Fall die **TextView text1**.
- Mapping des **title** Feldes des Cursors auf die **TextView text1** mittels zweier Arrays: String Array mit den Spalten Namen (Konstante **NotesDbAdapter.KEY_TITLE**) auf ein int Array mit den Referenzen der View an die die Daten gebunden werden sollen (**R.id.text1**).

```
private void fillData() {
    // Get all of the notes from the
    // database and create the item list
    Cursor c = mDbHelper.fetchAllNotes();
    startManagingCursor(c);

    String[] from = new String[] {
        NotesDbAdapter.KEY_TITLE };
    int[] to = new int[] { R.id.text1 };

    // Now create an array adapter and set
    // it to display using our row
    SimpleCursorAdapter notes = new
        SimpleCursorAdapter(
            this,
            R.layout.notes_row,
            c,
            from,
            to);
    setListAdapter(notes);
}
```

- **startManagingCursor** übergibt die Lifecycle Kontrolle für den Cursor an Android. (→ Übung 3)
- **from**, ein String Array mit Konstanten der Spaltennamen, **to**, das int-Array mit View-Id's
- **SimpleCursorAdapter** Instantiierung mit einem Kontext, dem Layout für die Zeilen der Liste (**notes_row**), den Cursor und den Arrays für das Mapping.
- Das Mapping zwischen **from** Spalten auf **to** Ressourcen kann auch mehr als eine Spalte abbilden und so mehrere Views an die Inhalte des Cursors binden.

```
private void fillData() {
    // Get all of the notes from the
    // database and create the item list
    Cursor c = mDbHelper.fetchAllNotes();
    startManagingCursor(c);

    String[] from = new String[] {
        NotesDbAdapter.KEY_TITLE };
    int[] to = new int[] { R.id.text1 };

    // Now create an array adapter and set
    // it to display using our row
    SimpleCursorAdapter notes = new
        SimpleCursorAdapter(
            this,
            R.layout.notes_row,
            c,
            from,
            to);
    setListAdapter(notes);
}
```

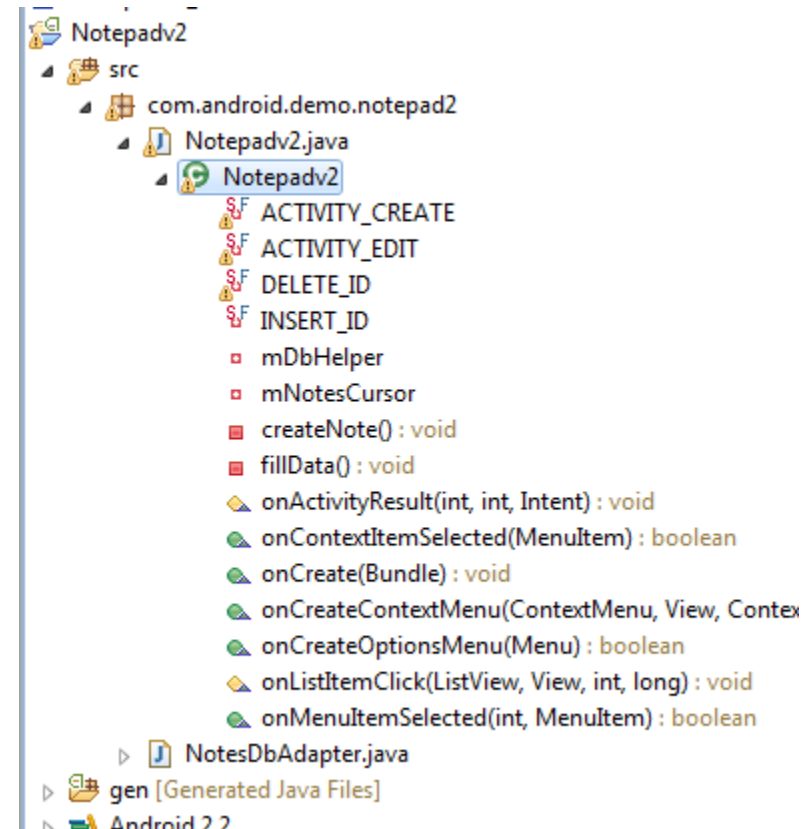
Übung 2

→ Lifiedemo der Ziel-Applikation

- Hinzufügen einer 2'ten Aktivität zur Applikation, mit der Notizen erzeugt und geändert werden können.
- Löschen einer Notiz über das Kontextmenü.
- Die neue Activity, ein Editor für Notizen, sammelt Benutzereingaben in einem **Bundle**, dass als Ergebnis des Aufrufs eines Intents zurückgegeben wird.
- **Lernziele und Inhalte:**
 - Erzeugen einer weiteren Aktivität und deren Konfiguration im Manifest
 - asynchroner Aufruf einer Aktivität mit Ergebnis
 - Daten zwischen Aktivitäten über **Bundle** Objekte austauschen
 - Verbessertes Bildschirmlayout kennenlernen
 - Kontextmenus verwenden.

Übung 2 – Schritt 1

- Android Projekt **Notepadv2** aus dem **Notepad CodeLab** erzeugen (analog Übung 1).
- Erweiterungen:
 - **strings.xml** in **res/values**: weitere Strings
 - Notepadv2 Klasse: neue Konstanten und das **mNotesCursor** Feld.
 - **fillData**: angepasst für **mNotesCursor**
 - neue überschriebene Methoden:
onCreateContextMenu(),
onContextItemSelected(),
onListItemClick() ,
onActivityResult()



Übung 2 – Schritt 1 – Kontextmenü für Listen

- Löschen von Items einer Liste über ein Kontextmenü → Registrieren der **List View** für das Kontextmenü mit **registerForContextMenu(getListView())** in **onCreate()**.
- **ListActivity** → **getListView()** liefert das **ListView** Objekt der Aktivität.
- **onCreateContextMenu()**: Callback Methode ähnlich zu dem Optionsmenü Callback → Konstruktion des Menüs analog.
- **onCreateContextMenu()** Parameter:
 - View, die das Menu triggert
 - Informationen über das selektierte Objekt enthält.
 - ➔ wichtig, bei mehr als einem Kontextmenü.

```
registerForContextMenu (
    getListView());
```

```
@Override
public void onCreateContextMenu (
    ContextMenu menu, View v,
    ContextMenuInfo menuInfo) {
    super.onCreateContextMenu (
        menu, v, menuInfo);
    menu.add(0,
        DELETE_ID,
        0,
        R.string.menu_delete);
}
```

- **onContextItemSelected()**: Wenn die ID des Menuitems DELETE_ID ist, soll die selektierte Notiz gelöscht werden.
- **AdapterContextMenuInfo getMenuInfo()** liefert ein Info Objekt, dessen **id** Feld der Zeilenidentifikation in der DB entspricht →

deleteNote(info.id) des NotesDb Adapter löscht die Note in der DB.

```
public boolean
onContextItemSelected(MenuItem item) {

    switch(item.getItemId()) {

        case DELETE_ID:
            AdapterContextMenuInfo info =
                (AdapterContextMenuInfo)
                    item.getMenuInfo();

            mDbHelper.deleteNote(info.id);
            fillData();
            return true;

    }

    return super.
        onContextItemSelected(item);
}
```

Übung 2 – Schritt 4 – Starten einer Aktivität

- **createNote** Methode erweitern:
 - Erzeugen eines **Intents**, um eine Notiz mit der **NoteEdit** Aktivität zu erzeugen (**ACTIVITY_CREATE**)
 - **Intent** mit **startActivityForResult()** feuern.
 - Übergabe eines Kontextes (**this**) für Zugriff zum Android Betriebssystem, um den Request zuzustellen
 - **startActivityForResult()** → Callback Methode **onActivityResult()** in der startenden Aktivität wird aufgerufen, wenn andere Aktivität endet.
- **startActivity()**: "fire-and-forget" Variante: startende Aktivität wird nicht über das Beenden der anderen informiert und erhält auch kein Ergebnis.

```
private void createNote() {
    Intent i = new Intent(this,
                          NoteEdit.class);
    startActivity(i,
                  ACTIVITY_CREATE);
}
```

Übung 2 – Schritt 5 – Notizen editieren

- Wird eine Notiz in der Liste selektiert, dann soll die Aktivität **NoteEdit** zum Editieren der Notiz (Aktion **ACTIVITY_EDIT**) mit den Daten der selektierten Notiz gestartet werden.
- Implementieren der **onListItemClick()** Methode mit den Parametern:
 - selektiertes **ListView** Objekth
 - selektierte View innerhalb der ListView
 - selektierte Position
 - **mRowId** des selektierten Items
- Über die Position Ermitteln der Daten der selektierten Zeile mit Hilfe des Cursors, die gebündelt und zur **NoteEdit** Aktivität geschickt werden.
- Die Methode erzeugt einen **Intent** für die **NoteEdit** Klasse und transportiert Daten in dem Extras Bundle des Intents. Übertragen werden Title and Body und die **mRowId** der Notiz.
- Starten mit **startActivityForResult()** und der Aktion **ACTIVITY_EDIT**.

Übung 2 – Schritt 5 – Notizen bearbeiten

```
@Override
protected void onItemClick(ListView l, View v, int position, long id) {
    super.onItemClick(l, v, position, id);
    Cursor c = mNotesCursor;
    c.moveToPosition(position);
    Intent i = new Intent(this, NoteEdit.class);

    i.putExtra(NotesDbAdapter.KEY_ROWID, id);
    i.putExtra(NotesDbAdapter.KEY_TITLE,
        c.getString(
            c.getColumnIndexOrThrow(NotesDbAdapter.KEY_TITLE)));
    i.putExtra(NotesDbAdapter.KEY_BODY, c.getString(
        c.getColumnIndexOrThrow(NotesDbAdapter.KEY_BODY)));

    startActivityForResult(i, ACTIVITY_EDIT);
}
```

- Weitere Erläuterungen
 - **putExtra()** Einfügen von Elementen in ein **Bundle**, das beim Aufruf transportiert wird.
 - Details der Notiz werden aus dem Cursor gelesen, der an die Position des selektierten Elementes gesetzt wird (**moveToPosition()**).
 - Aufruf des **Intents** mit dem Extras auf der Klasse **NoteEdit** mit **startActivityForResult()**
 - Übergabe des Requestcodes **ACTIVITY_EDIT**: Requestcode wird der Methode **onActivityResult** in der aufrufenden Aktivität übergeben und erlaubt die Zuordnung von Aufruf zu den Ergebnissen.

Übung 2 – Schritt 6 – Ergebnisse bearbeiten

- **createNote** und **onListItemClick** verwenden asynchronen Intent Aufruf.
- **onActivityResult**: asynchroner Call-back Handler für das Verarbeiten von Ergebnissen
- Parameter:
 - **requestCode** — Requestcode des Intent Aufrufs (**ACTIVITY_CREATE** oder **ACTIVITY_EDIT**).
 - **resultCode** — Ergebnis- oder Fehlercode (0 für korrektes Ergebnis, sonst nicht korrektes Ergebnis)
 - **intent** — der **Intent**, der von der Aktivität, die das Ergebnis erzeugt hat, geschickt wird.
- Die Kombination aus **startActivityForResult()** und **onActivityResult()** entspricht dem asynchronen RPC (remote procedure call)
- Beste Technik für Aktivitäten sich gegenseitig aufzurufen und um to invoke another and share services.

Übung 2 – Schritt 6 – Ergebnisse bearbeiten

- **createNote** und **onListItemClick** verwenden asynchronen Intent Aufruf.
- **onActivityResult**: asynchroner Callback Handler für das Verarbeiten von Ergebnissen
- Parameter:
 - **requestCode** — Requestcode des Intent Aufrufs (**ACTIVITY_CREATE** oder **ACTIVITY_EDIT**).
 - **resultCode** — Ergebnis- oder Fehlercode (0 für korrektes Ergebnis, sonst nicht korrektes Ergebnis)
 - **intent** — der **Intent**, der von der Aktivität, die das Ergebnis erzeugt hat, geschickt wird.
- Die Kombination aus **startActivityForResult()** und **onActivityResult()** entspricht dem asynchronen RPC (remote procedure call)
- Beste Technik für Aktivitäten sich gegenseitig aufzurufen und um Dienste zu teilen.

Übung 2 – Schritt 6 – Ergebnisse bearbeiten

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);
    Bundle extras = intent.getExtras();

    switch (requestCode) {
        case ACTIVITY_CREATE:
            String title = extras.getString(NotesDbAdapter.KEY_TITLE);
            String body = extras.getString(NotesDbAdapter.KEY_BODY);
            mDbHelper.createNote(title, body);
            fillData();
            break;
        case ACTIVITY_EDIT:
            Long mRowId = extras.getLong(NotesDbAdapter.KEY_ROWID);
            if (mRowId != null) {
                String editTitle = extras.getString(NotesDbAdapter.KEY_TITLE);
                String editBody = extras.getString(NotesDbAdapter.KEY_BODY);
                mDbHelper.updateNote(mRowId, editTitle, editBody);
            }
            fillData();
            break;
    }
}
```

Übung 2 – Schritt 7 – Layout für NoteEdit

- ***note_edit.xml***: → UI Layout für den Notizeneditor
- ***android:layout_weight***: in ***Linear Layouts*** verwendet, um Views untereinander zu gewichten. Der Anteil des Gewichts am Gesamtgewicht bestimmt den Bereich des Bildschirms, den eine View einnimmt. Default 0: View belegt soviel Platz, wie die enthaltenen Komponenten benötigen. Werte > 0 teilen den Bildschirm der Parent View in Bereiche, die den Gewichten der enthaltenen Views entsprechen.
- **Einbettungen**: ein horizontales lineares Layout ist in ein vertikales Layout eingebettet, um Titel und Text der Notiz horizontal auszurichten.

Übung 2 – Schritt 7 – Layout für NoteEdit

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent" ...>

    <LinearLayout android:orientation="horizontal" android:layout_width="match_parent"
        ...>

        <TextView android:layout_width="wrap_content" ... android:text="@string/title" />
        <EditText android:id="@+id/title" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:layout_weight="1"/>
    </LinearLayout>

    <TextView android:layout_width="wrap_content" ... android:text="@string/body" />
    <EditText android:id="@+id/body" android:layout_width="match_parent" ...
        android:layout_weight="1" android:scrollbars="vertical" />

    <Button android:id="@+id/confirm"
        android:text="@string/confirm" android:layout_width="wrap_content" ... />

</LinearLayout>
```

Übung 2 – Schritt 8 – NoteEdit Aktivität

- Klasse **NoteEdit** erzeugen und von von **android.app.Activity** ableiten
- **onCreate()** überschreiben

```
public class NoteEdit extends Activity {  
  
    @Override  
    protected void onCreate(  
        Bundle savedInstanceState) {  
        super.onCreate(  
            savedInstanceState);  
        }  
    }  
}
```

Übung 2 – Schritt 9 – NoteEdit onCreate

- Titel der Aktivität setzen → "Edit Note"
- Layout der Aktivität setzen → Content View wird zu *note_edit.xml*
- Referenzen auf View-Objekte:
 - *title* und *body* TextEdit Views
 - *confirmButton*
- Auslesen der Werte aus dem *Intent Bundle*
 - Setzen der Inhalte der *title* und *body* TextEdit Views
 - Lesen und Speichern der *mRowId*, die Referenz auf die editierte Notiz

```
public class NoteEdit extends Activity {

    private EditText mTitleText;
    private EditText mBodyText;
    private Long mRowId;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.note_edit);
        setTitle(R.string.edit_note);
        mTitleText = (EditText)
            findViewById(R.id.title);
        mBodyText = (EditText)
            findViewById(R.id.body);
        Button confirmButton = (Button)
            findViewById(R.id.confirm);
    }
}
```

Übung 2 – Schritt 9 – NoteEdit onCreate

- Titel der Aktivität setzen → "Edit Note"
- Layout der Aktivität setzen → Content View wird zu *note_edit.xml*
- Referenzen auf View-Objekte:
 - *title* und *body* EditText Views
 - *confirmButton*
- Auslesen der Werte aus dem *Intent Bundle*
 - Setzen der Inhalte der *title* und *body* EditText Views
 - Lesen und Speichern der *mRowId*, die Referenz auf die editierte Notiz

```
public class NoteEdit extends Activity {

    ...

    @Override
    protected void onCreate(Bundle
        savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        mRowId = null;
        Bundle extras = getIntent().
            getExtras();

        if (extras != null) {
            String title = extras.getString(
                NotesDbAdapter.KEY_TITLE);
            String body = extras.getString(
                NotesDbAdapter.KEY_BODY);
            mRowId = extras.getLong(
                NotesDbAdapter.KEY_ROWID);
            if (title != null) {
                mTitleText.setText(title);
            }
            if (body != null) {
                mBodyText.setText(body);
            }
        }
    }
}
```

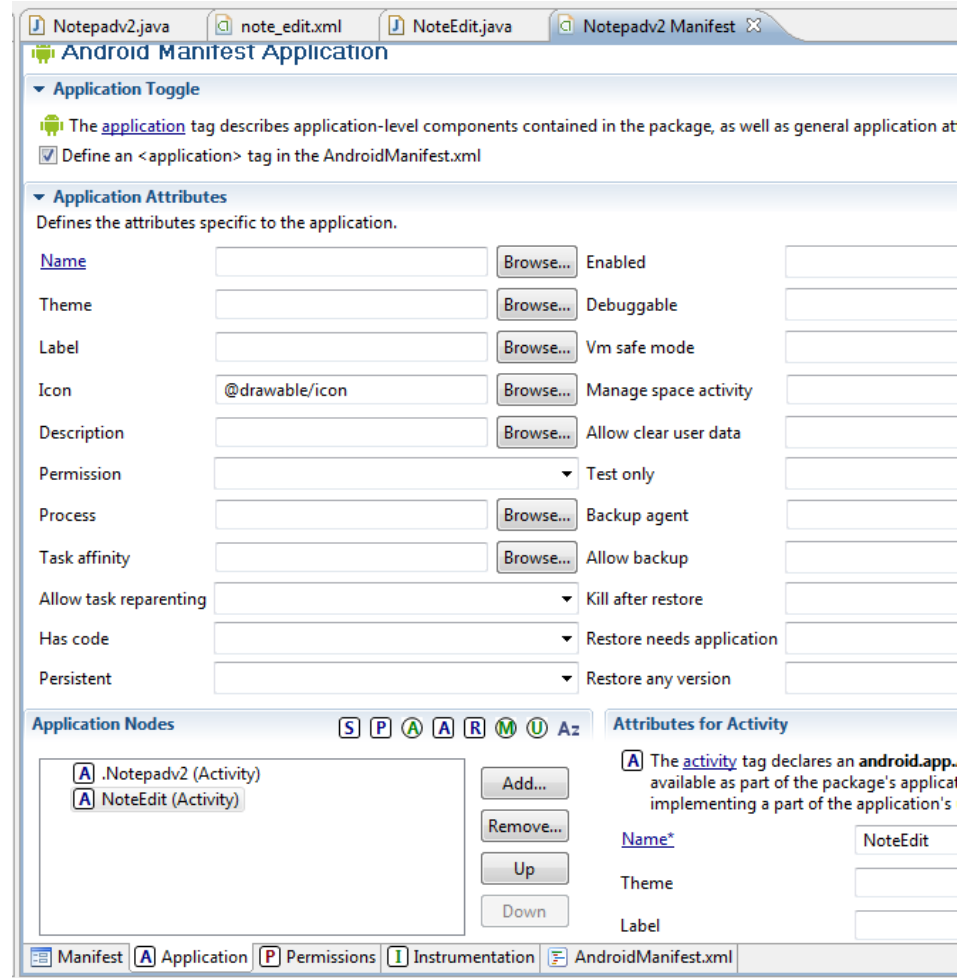
Übung 2 – Schritt 10 – NoteEdit OnClickListener

- Ein Klick auf den **confirmButton** soll
 - Änderungen in den Views einsammeln
 - Änderungen in ein Bundle eintragen
 - Bei einer Editieraktion mRowId in das Bundle eintragen
 - Ergebnis Intent erzeugen und Ergebnis setzen
 - Aktivität beenden
- **setResult():**
 - setzt Ergebnis Code (z.B. **RESULT_OK**)
 - setzt den Intent, der an den Intent Aufrufer zurückgesendet wird.
- **finish()**
 - signalisiert das Beenden einer Aktivität
 - Ergebnis und Ausführungskontrolle gehen an die aufrufende Aktivität über

```
confirmButton.setOnClickListener(  
    new View.OnClickListener() {  
  
        public void onClick(View view) {  
            Bundle bundle = new Bundle();  
            bundle.putString(  
                NotesDbAdapter.KEY_TITLE,  
                mTitleText.getText().toString()  
            );  
            bundle.putString(  
                NotesDbAdapter.KEY_BODY,  
                mBodyText.getText().toString());  
            if (mRowId != null) {  
                bundle.putLong(  
                    NotesDbAdapter.KEY_ROWID,  
                    mRowId);  
            }  
            Intent mIntent = new Intent();  
            mIntent.putExtras(bundle);  
            setResult(RESULT_OK, mIntent);  
            finish();  
        }  
    });
```

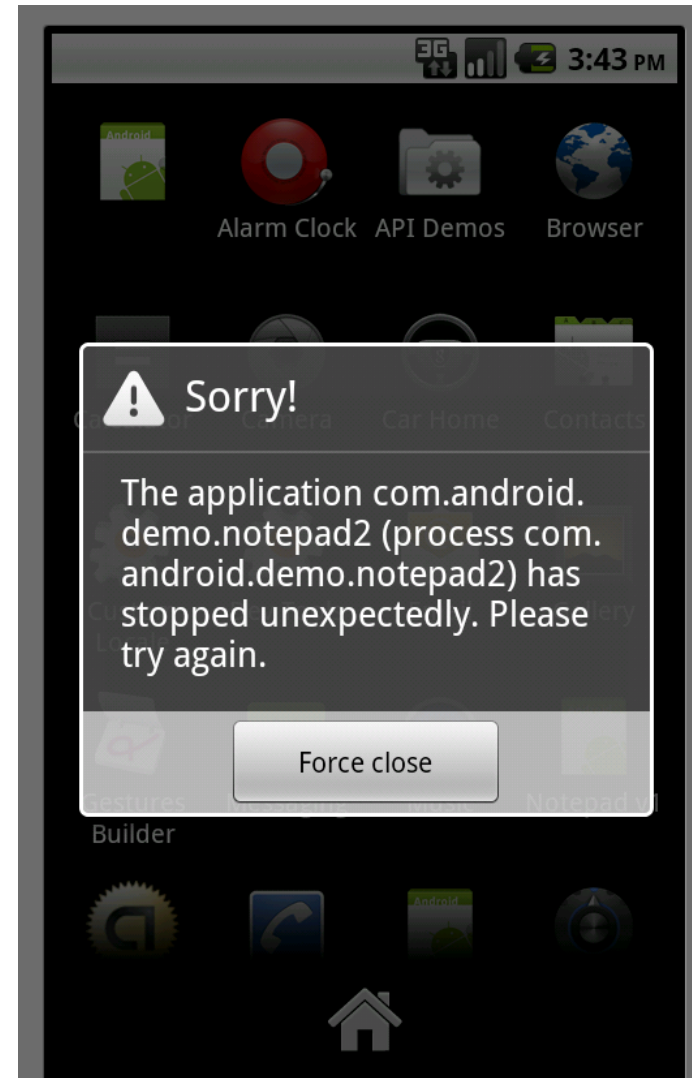
Übung 2 – Schritt 11 – NoteEdit registrieren

- Registrieren der Aktivität im **Android Manifest.xml** meldet die Aktivität beim System an.
- Ohne Registrierung kennt das System die Aktivität nicht und kann auch keine Intents zu-
stellen.
- Nutzen des Manifest-Editors
 - Öffnen mit Doppelklick auf die Manifest-Datei
 - Reiter Application > **Add** (im Bereich Application Nodes)
 - **NoteEdit**, Klassenname der Aktivität eintragen



Übung 2 – Schritt 12 – Installieren und Testen

- Applikation **Notepadv2** auf einem virtuellen Device installieren.
- Eine Notiz editieren
- den “back” Button des Emulators statt des “confirm” Buttons verwenden.
 - Erzeugt einen Fehler
 - Alle Änderungen werden verworfen
- ➔ Lösung in Übung 3
- ➔ Korrekte Behandlung des Lifecycles der **NoteEdit** Aktivität.



- **Inhalte:**
 - Verwendung von Lifecycle Events
 - Techniken, um den Zustand einer Applikation zu verwalten
- **Notepadv3** importieren
- **NoteEdit onCreate:**
 - Zeilen, in denen **title** und **body** aus dem Bundle gelesen werden und die zugehörigen Abfragen löschen.
 - **später:** ersetzt durch Abfragen an die Datenbank über den **NotesDBAdapter**

```
String title = extras.getString(  
    NotesDbAdapter.KEY_TITLE);  
String body = extras.getString(  
    NotesDbAdapter.KEY_BODY);  
  
if (title != null) {  
    mTitleText.setText(title);  
}  
  
if (body != null) {  
    mBodyText.setText(body);  
}
```

Übung 3 – Schritt 2

- Vorbereitungen für den Zugriff auf die Notizen Datenbank:
 - Klassenvariable für den **NotesDbAdapter** in der **NoteEdit** Klasse einführen
 - Instanz des **NotesDbAdapter** in der **onCreate()** Methode erzeugen und DB öffnen

```
public class NoteEdit extends Activity {

    ...

    private NotesDbAdapter mDbHelper;

    @Override
    protected void onCreate(Bundle
                               savedInstanceState) {

        super.onCreate(
            savedInstanceState);

        mDbHelper = new NotesDbAdapter(
                               this);

        mDbHelper.open();

        ...

    }

    ...

}
```

Übung 3 – Schritt 3

- Zu überprüfen ist, ob eine Notiz editiert wird:

→ **savedInstanceState** oder Bundle enthält eine **mRowId**

- **savedInstanceState != null**

→ “saved state” der Aktivität muss wieder hergestellt werden (für den Fall, dass die Aktivität den Fokus verloren hat und neu gestartet wird).

→ Lesen von **mRowId** aus **savedInstanceState**

- **mRowId** danach noch **null**:

→ Lesen von **mRowId** aus dem Bundle

- **mRowId** weder in **savedInstanceState** noch im **Bundle**

→ es wird eine neue Notiz erzeugt.

```
mRowId = null;
Bundle extras = getIntent().getExtras();
if (extras != null) {
    mRowId = extras.getLong(
        NotesDbAdapter.KEY_ROWID);
}
```

ersetzen durch



```
mRowId =
    (savedInstanceState == null) ? null :
    (Long) savedInstanceState.
        getSerializable(
            NotesDbAdapter.KEY_ROWID);
if (mRowId == null) {
    Bundle extras = getIntent().
        getExtras();

    mRowId =
        extras != null ?
        extras.
            getLong(
                NotesDbAdapter.KEY_ROWID)
            : null;
}
```

- Views von **NoteEdit** mit Inhalten befüllen:
 - Lesen der Daten zu einer **mRowId** über den **NotesDbAdapter** (Edit Modus)
 - Erzeugen einer neuen **mRowId** über den **NotesDbAdapter** (Create Modus)
- Methode → **populateFields**
 - Aufruf einfügen vor der Registrierung des **OnClickListeners**

Übung 3 – Schritt 5

- **Idee:**

- Notizen der **NoteEdit** Aktivität werden gespeichert, wenn eine Aktivität normal beendet oder vom System “entfernt” wird.
- Zustand “EDIT” wird gespeichert, wenn eine Aktivität vom System “entfernt” wird.

→ **NoteEdit** muss die Daten nicht mehr an die aufrufende Aktivität übertragen

- **onClick** Methode ruft nur die Kurzform **setResult(RESULT_OK)** auf

```
confirmButton.setOnClickListener(new
    View.OnClickListener() {
        public void onClick(View view) {
            setResult(RESULT_OK);
            finish();
        }
    });
```

- **populateFields** Methode

- wenn **mRowId** definiert ist, werden die Daten aus der DB gelesen und in die **EditText** Views übertragen.

- **startManagingCursor()**:

- Methode der Klasse **Activity**
- verwaltet den Cursor Lifecycle (Freigabe und Wiederherstellung) abhängig vom Activity Lifecycle

```
private void populateFields() {
    if (mRowId != null) {
        Cursor note = mDbHelper.fetchNote(
                                                                    mRowId);
        startManagingCursor(note);

        mTitleText.setText(
            note.getString(
                note.getColumnIndexOrThrow(
                    NotesDbAdapter.KEY_TITLE)));

        mBodyText.setText(
            note.getString(
                note.getColumnIndexOrThrow(
                    NotesDbAdapter.KEY_BODY)));
    }
}
```

Übung 3 – Schritt 7+8 – Lifecycle und Zustandssicherung

- Überschreiben der Lifecycle Methoden
 - ***onSaveInstanceState***, ***onPause***,
onResume
- ***onSaveInstanceState***:
 - wird aufgerufen wenn eine Aktivität gestoppt wird und vor der Wiederherstellung gelöscht werden könnte.
 - Um den Zustand der Aktivität beim Neustart rekonstruieren zu können, muss dieser als Bundle gespeichert werden.
 - Zustand wird als ***savedInstanceState*** Bundle beim Aufruf von onCreate übergeben.
- ***onPause*** / ***onResume***:
 - ***onPause()*** wird aufgerufen, wenn eine Aktivität endet. In diesem Zustand wird die aktuell bearbeitete Notiz in die Datenbank geschrieben.
 - ***onResume()*** verwendet dann die ***populateFields()*** Methode um Notizen aus der Datenbank zu lesen.

Übung 3 – Schritt 7 + 8

Lifecycle und Zustandssicherung

```
@Override
```

```
protected void onSaveInstanceState(Bundle  
    outState) {
```

```
    super.onSaveInstanceState(outState);  
    saveState();  
    outState.putSerializable(  
        NotesDbAdapter.KEY_ROWID,  
        mRowId);
```

```
}
```

```
@Override
```

```
protected void onPause() {  
    super.onPause();  
    saveState();  
}
```

```
@Override
```

```
protected void onResume() {  
    super.onResume();  
    populateFields();  
}
```

```
private void saveState() {
```

```
    String title =  
        mTitleText.getText().toString();  
    String body =  
        mBodyText.getText().toString();
```

```
if (mRowId == null) {
```

```
    long id =  
        mDbHelper.createNote(  
            title, body);
```

```
    if (id > 0) {  
        mRowId = id;
```

```
    }
```

```
} else {  
    mDbHelper.updateNote(  
        mRowId, title, body);
```

```
}
```

```
}
```

- ***onActivityResult()*** in ***Notepadv3***:

→ Notizen lesen und aktualisieren in der ***NoteEdit*** Klasse

→ ***Notepadv3*** muss nur noch den Inhalt der Liste aktualisieren und kein Ergebnisbundle auslesen.

```
@Override
protected void onActivityResult(
    int requestCode, int resultCode,
        Intent intent) {
    super.onActivityResult(
        requestCode, resultCode,
            intent);

    fillData();
}
```

- ***onListItemClick()*** Methode

→ Zeilen die ***title*** und ***body*** aus der Datenbank lesen und in das Bundle schreiben können entfernt werden

→ nur die ***mRowId*** wird benötigt

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

    Intent i = new Intent(this,
        NoteEdit.class);

    i.putExtra(
        NotesDbAdapter.KEY_ROWID, id);
    startActivityForResult(i,
        ACTIVITY_EDIT);
}
```

- Nutzen des Eclipse Debuggers um die Lifecycle Methoden zu verfolgen.