ITSMAP F13 Lesson 9

ContentProvider

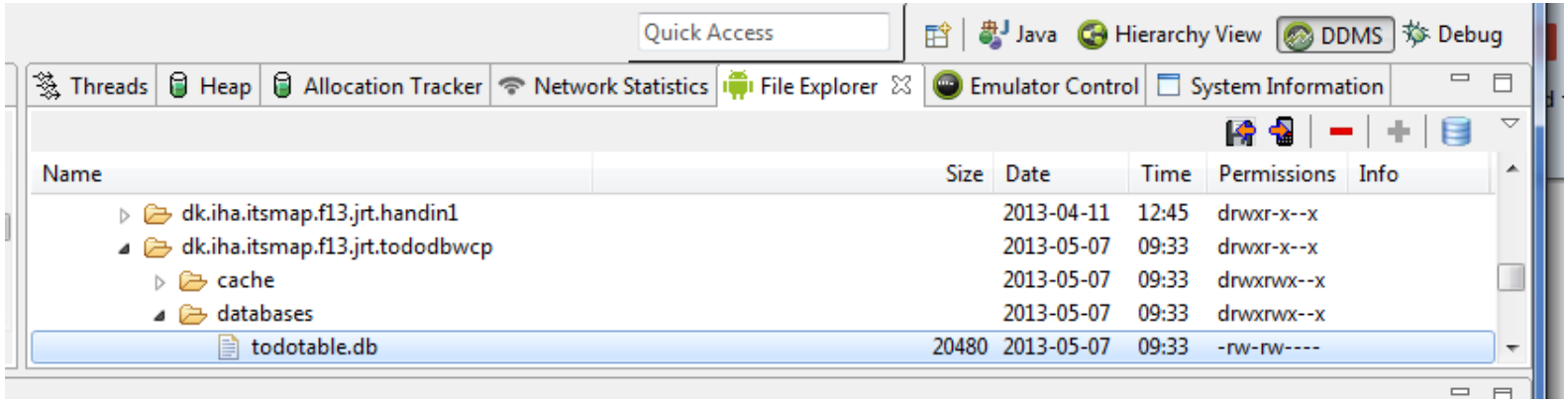
Jesper Rosholm Tørresø

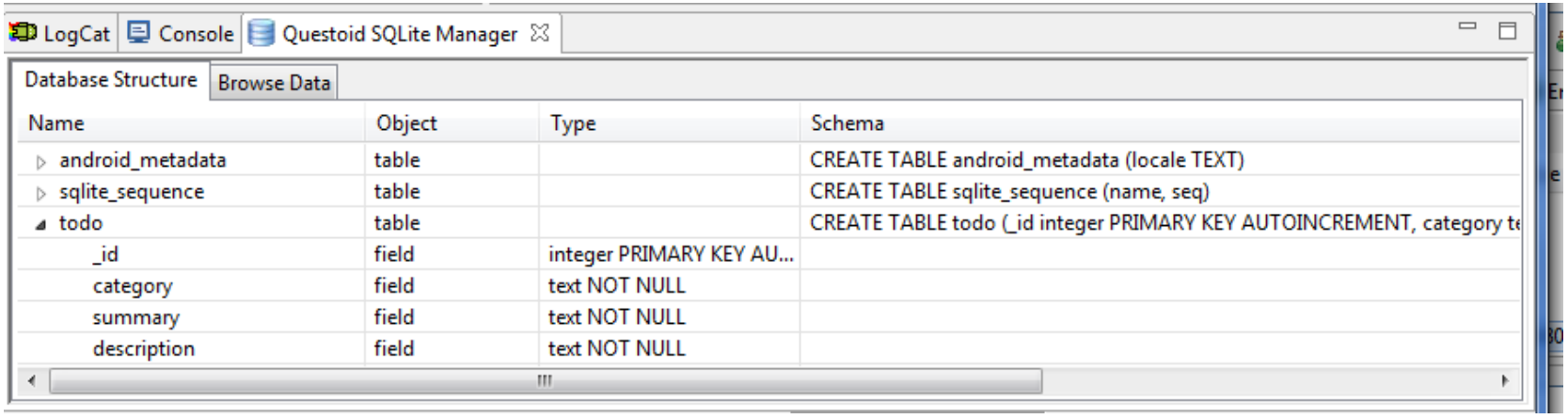
This lesson

- Midtvejsevaluering 15 min.
- Preferences/SQLite follow up
- Short introduction to ContentProviders
- Block 2-4 Exercise 1+2+3

SQLite Plugin for Eclipse

- http://www.coderzheaven.com/2011/04/18/sqlitemanager-plugin-for-eclipse/





Query the SQLite RDBMS

- **rawQuery() versus query()**

```
Cursor cursor = getReadableDatabase().
  rawQuery("select * from todo where _id = ?", new String[] { id });
```

```
return database.query(DATABASE_TABLE,
  new String[] { KEY_ROWID, KEY_CATEGORY, KEY_SUMMARY, KEY_DESCRIPTION },
  null, null, null, null, null);
```

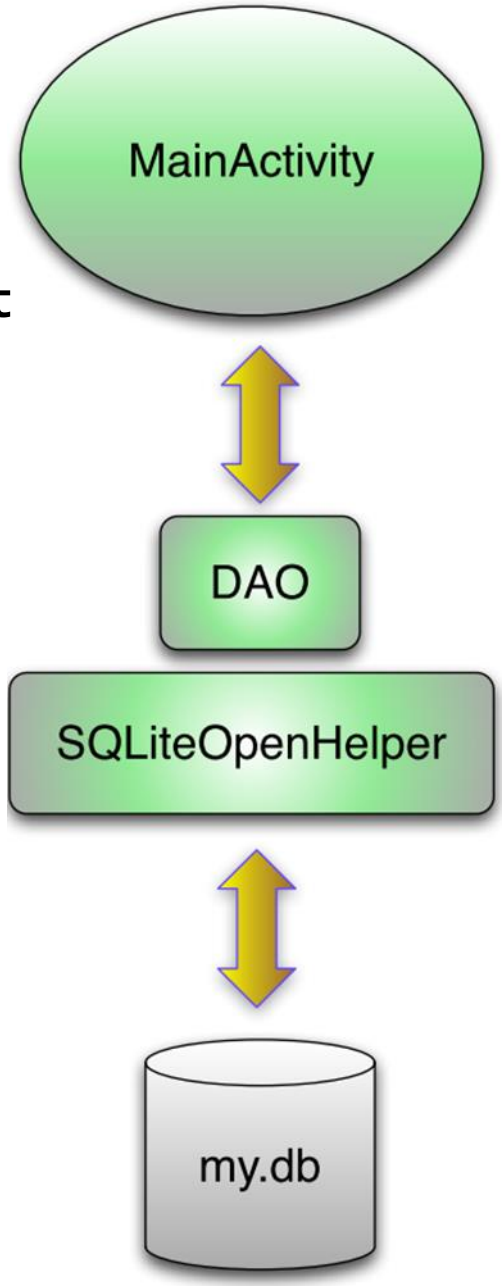
Decompose query for the query() method

Parameter	Comment
String dbName	The table name to compile the query against.
String[] columnNames	A list of which table columns to return. Passing "null" will return all columns.
String whereClause	Where-clause, i.e. filter for the selection of data, null will select all data.
String[] selectionArgs	You may include ?s in the "whereClause"". These placeholders will get replaced by the values from the selectionArgs array.
String[] groupBy	A filter declaring how to group rows, null will cause the rows to not be grouped.
String[] having	Filter for the groups, null means no filter.
String[] orderBy	Table columns which will be used to order the data, null means no ordering.

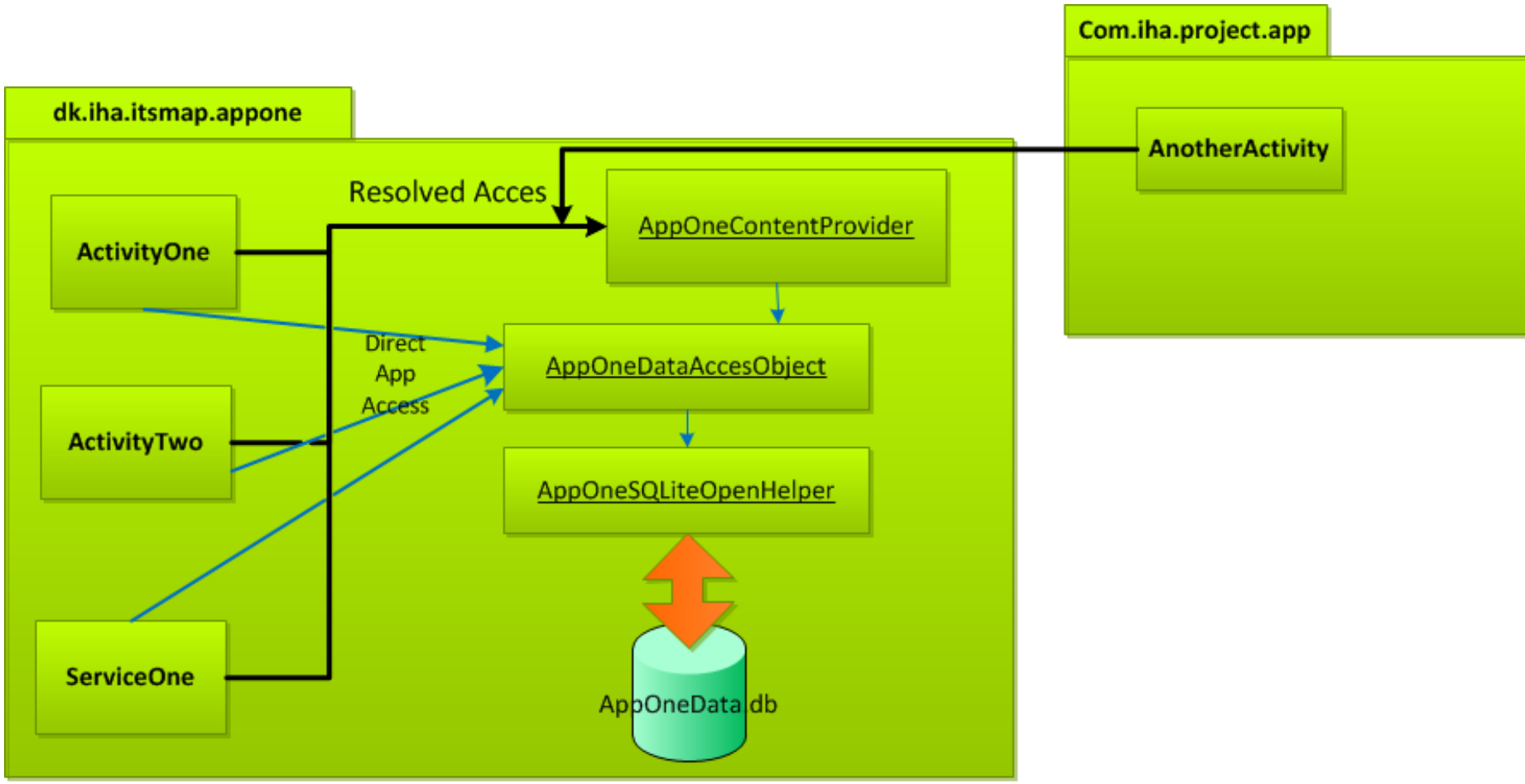
Architecture SQLite

Classes to deal with

- **`dk.iha.itsmap.appone.AppOneDataAccessObject`**
  - Your own adapter for CRUD operations on the database
- **`android.database.sqlite.SQLiteOpenHelper`**
  - Your own helper Class for creating, opening and upgrading the database (Abstract class, you implement the App specific part)
- **`android.database.sqlite.SQLiteDatabase`**
  - SQLite Database driver
- **`android.database.Cursor;`**
  - A Cursor implementation that exposes results from a query on a SQLiteDatabase. (AKA a result set) .



DB Arch. + ContentProvider




```
private static String DB_PATH = "/data/data/dk.iha.itsmap.appone/databases/";
```

Android Framework

Developers Guide ContentProvider

- http://developer.android.com/guide/topics/providers/content-provider-basics.html

ContentProvider
Steps for resolved request

- A ContentProvider Acts in a way like “a device local RESTFul host”
- Register your ContentProvider to the Android FW in the Manifest
- The “authorities” identifies your ContentProvider and it is resolved by this value!

```
<provider
android:name=".MyContentProvider"
android:authorities="dk.iha.itsmap.themeapp.mycontentprovider" />
```

Access and Permissions

```
<provider
      android:name=".MyTodoContentProvider“
      android:authorities="dk.iha.itsmap.f13.jrt.todos.contentprovider"
      android:enabled="true"
      android:exported="true"
      android:permission="dk.iha.itsmap.F13.READWRITE_TODO"
      android:readPermission="dk.iha.itsmap.F13.READONLY_TODO"
      android:writePermission="dk.iha.itsmap.F13.WRITE_TODO" >
 </provider>

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

<uses-permission android:name="dk.iha.itsmap.F13.READWRITE_TODO“ />
```

http://developer.android.com/guide/topics/manifest/provider-element.html

URI’ are used for specifying the request type

- General URI form

content://<domaincode>.<company>.provider.<appname/<datapath>

Ex. Without parameter

content://dk.iha.itsmap.provider.themeapp/items

Ex. With parameter

content://dk.iha.itsmap.provider.themeapp/items/67

ContentProvider
CRUD on persistent data

The primary methods that need to be implemented are:

- onCreate() which is called to initialize the provider
- query(Uri, String[], String, String[], String) which returns data, Cursor, to the caller
- insert(Uri, ContentValues) which inserts new data into the content provider
- update(Uri, ContentValues, String, String[]) which updates existing data in the content provider
- delete(Uri, String, String[]) which deletes data from the content provider
- getType(Uri) which returns the Multipurpose Internet Mail Extensions, MIME type, of data in the content provider

Click links for description of each method

URIs needs to be parsed in the ContentProvider

```
public class MyContentProvider extends ContentProvider {
public static final Uri CONTENT_URI = Uri.parse
        ("content://dk.iha.itsmap.themeapp.mycontentprovider");
…

// ** Using the UriMatcher to handle single or multiple query requests
// Create the constants used to differentiate between the different URI
// requests.
private static final int ALLROWS = 1;
private static final int SINGLE_ROW = 2;

private static final UriMatcher uriMatcher;

// Populate the UriMatcher object, where a URI ending in 'items' will
// correspond to a request for all items, and 'items/[rowID]'
// represents a single row.
static {
uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
uriMatcher.addURI("dk.iha.itsmap.themeapp.mycontentprovider", "items", ALLROWS);
uriMatcher.addURI("dk.iha.itsmap.themeapp.mycontentprovider", "items/#",
SINGLE_ROW);
}
```

URIs needs to be parsed in the ContentProvider

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {
// If this is a row query, limit the result set to the passed in row.
  switch (uriMatcher.match(uri)) {
    case SINGLE_ROW :
      // TODO: Modify selection based on row id, where:
      // rowNumber = uri.getPathSegments().get(1));
  }
return null;
}

public int delete(Uri uri, String selection, String[] selectionArgs) {
switch (uriMatcher.match(uri)) {
    case ALLROWS: return 0;  //return rows deleted
    case SINGLE_ROW: return 0; //return rows deleted
    default: throw new IllegalArgumentException("Unsupported URI:" +
                                                uri);
  }
}
```

ContentProvider
MIME types /Content types

```
@Override
public String getType(Uri uri) {
        switch (uriMatcher.match(uri)) {
        case ALLROWS:
        return "vnd.android.cursor.dir/myprovidercontent";
        case SINGLE_ROW:
        return "vnd.android.cursor.item/myprovidercontent";
        default:
        throw new IllegalArgumentException("Unsupported URI: " +
uri);
        }
}
```

Requesting the ContentProvider "The straight forward way"

```
Public class Tester extends  Activity {//extends Service
private static final String KEY_COL3 = "UserName";
private static final String KEY_COL5 = "asc";
    public void CallCP()
    {
        // ** Listing 7-13: Querying a Content Provider with a Content Resolver
        ContentResolver cr = getContentResolver();

        // Return all rows
        Cursor allRows = cr.query(MyContentProvider.CONTENT_URI, null, null,
        null, null);
        String requiredValue ="17";
        // Return all columns for rows where column 3 equals a set value
        // and the rows are ordered by column 5.
        String where = KEY_COL3 + "=" + requiredValue ;
        String order = KEY_COL5;
        Cursor someRows = cr.query(MyContentProvider.CONTENT_URI, null, where,
        null, order);
    }
}
```

Querying a ContentProvider

- Retrieving data can be a time consuming function to proceed.
- Activity and Fragment life cycle may interrupt setting up a Cursor (the result set from CP)
- You must set the query into background and manage the Cursor to maintain retrieved data
- Two approaches
  - Use managedQuery () < API L10 comes from Activity
  - Use CursorLoader => API L11 goes for Fragment and Activtity

Querying a ContentProvider

- managedQuery *From API L11 Depricated*
  - **managedQuery()** will use ContentResolver's **query()**. The difference is that with managedQuery() the activity will keep a reference to your Cursor and close it whenever needed (in onDestroy() for instance.) **If you do query() yourself, you will have to manage the Cursor as a sensitive resource.** If you forget, for instance, to close() it in onDestroy(), you will leak underlying resources **(logcat will warn you about it.)**
- CursorLoader (HoneyComb->)

CusorLoader

```
public class TodoOverviewActivity extends ListActivity implements
LoaderManager.LoaderCallbacks<Cursor> {
……..
getLoaderManager().initLoader(0, null, this); //Setup CusorLoader
……..
// Creates a new loader after the initLoader () call
@Override
  public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    String[] projection = { TodoTable.COLUMN_ID, TodoTable.COLUMN_SUMMARY };
    CursorLoader cursorLoader = new CursorLoader(this,
        MyTodoContentProvider.CONTENT_URI, projection, null, null, null);
    return cursorLoader;
  }
  @Override
  public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    adapter.swapCursor(data);
  }
  @Override
  public void onLoaderReset(Loader<Cursor> loader) {
    // data is not available anymore, delete reference
    adapter.swapCursor(null);
  }
```

Exercise 1

- Create an application to add customers and show customer data: name and address
- Use a DAO and a SQLiteOpenHelper.
- Implement insert, delete(id) and find(id)
- What happens when you use TEXT type data in a INTEGER FIELD ?

http://www.vogella.com/articles/AndroidSQLite/article.html

Exercise 2 Make a ContentProvider

- Make a ContentProvider to the customer database from exercise 1
- Use the different templates available
- Make an Activity accessing the database using the ContentProvider
- This exercise is the last Hand In 5!

Exercise 3
Use Contacts ContentProvider

- You must create a new project
- Create at least 10 contacts on your phone or emulator.
- The application must query the contacts on the phone for an id (ContactsContract.Contacts._ID) and the contact name (ContactsContract.Contacts.DISPLAY_NAME) using the content provider and display the contact names in a list view in the application.
- When clicking a contact name you must display the id of the contact you clicked.