CS 522—Spring 2015
Mobile Systems and Applications
Assignment Four—Content Providers

All of your projects for this submission should target Lollipop (Android 5.0.1).

Part 1: Book Store

In the previous assignment, you implemented a book store app that stored the contents of the shopping cart in a SQLite database. In this assignment, you will provide a similar app, but with the shopping cart instead stored in a content provider. This content provider should provide a single table, identified by the content path, `books`. Internally, the content provider will store the data using a SQLite database with two tables, one for the books and the other for authors. As before, assume a one-to-many relationship from books to authors, and assume eager retrieval of authors for a book, as a synthetic column in the result. You should again have subactivities for adding a book to the cart, for clearing the cart, and for viewing the details of a book (including seeing all authors). The last activity is started when the user presses the line for a book in the main list view. A long press (at least two seconds) should place the book activity into contextual action mode, where the user may select books for deletion, and the contextual action bar provides a single DELETE action. Allow multi-item selection: the user can select any number of books for deletion (not just a single book) before choosing the DELETE action.

As another addition to the previous assignment, the main book activity list view lists all authors of a book in the second line of its list view entry, separated by the separator character. Fix this for this assignment, by defining a custom cursor adapter that extracts the first author name from the list of book authors and just displays that. Do this by extending `ResourceCursorAdapter`, and specializing the `bindView()` method that binds the fields in the view for a line of the list view:

```
public class BookAdapter extends ResourceCursorAdapter {

  protected final static int ROW_LAYOUT = android.R.layout.simple_list_item_2;

  public BookAdapter(Context context, Cursor cursor) {
    super(context, ROW_LAYOUT, cursor, 0);
  }

  @Override
  public View newView(Context context, Cursor cur, ViewGroup parent) {
    LayoutInflater inflater = (LayoutInflater)
       context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    return inflater.inflate(ROW_LAYOUT, parent, false);
  }

  @Override
  public void bindView(View view, Context context, Cursor cursor) {
    TextView titleLine = (TextView) view.findViewById(android.R.id.text1);
    TextView authorLine = (TextView) view.findViewById(android.R.id.text2);
```

```
    // etc
  }
}
```

For book detail activity, you can simply use an `ArrayAdapter` to display the names of the authors of a book, but do not do this for the main book activity where you display all books.

You should provide two solutions to this exercise.

Call the first solution `BookstoreWithContentProvider`. You should follow these guidelines for this solution:

1. Define a contract, `BookContract`, for the content provider, `BookProvider`. Place the former in the `contracts` subpackage of your app, and the latter in the `providers` subpackage of your app. This is a practice that you will be expected to follow for all assignments for the remainder of the course. The contracts class should define content URIs and content paths, content types, the column names for the cursor and content values, and operations for retrieving columns from a cursor and inserting them into a content values table.
2. Define an entity class, `Book`, for book entities stored in the database. You should have done most of this for the previous assignment. This should define entity fields, an implementation of the `Parcelable` interface, a constructor for initializing a book entity from a cursor, and an operation `writeToProvider` for initializing a `ContentValues` object (for insertion into a provider) with the fields of the entity.
3. All query operations should be asynchronous (performed on a background thread, never on the main UI thread). For queries that populate the UI in the main book activity or the book detail activity, use cursor loaders and the loader manager to query the provider. For the main book activity, it is important that you reuse the results of earlier queries on the loader manager. For the book detail activity, it is important that you **not** reuse the results of earlier queries on the loader manager (Why?). *Define all callbacks for the loader manager as methods in your activities, not in separate callback objects.* Note that this means that you must subclass `Activity` rather than `ListActivity` for the main books activity.

Call the second solution `BookstoreWithEntityManager`. This follows the first two guidelines for the solution above: define a contract, provider and entity class. However follow these guidelines instead of the third criteria for the solution above:

3. **All** content provider operations should be asynchronous (not just queries). Furthermore, all access to the provider by the application should be defined through a manager object, `BookManager`, that extends a generic abstract class `Manager<T>` as defined in the lecture materials (instantiating it with the `Book` entity type). The manager base class should define both synchronous and asynchronous content resolvers, and should be defined in the `managers` subpackage with the `BookManager` class. You will have to define the latter class, `AsyncContentResolver`, inheriting from `AsyncQueryHandler`. The manager base class should define generic factory methods for asynchronous queries (both using loaders and using the asynchronous content resolver). The `BookManager` class should define whatever app-specific type-

safe operations are required for the app to use the content provider, without accessing it directly. For loader queries, define a `TypedCursor<T>` class in the managers package that encapsulates a cursor and provides a type-safe API for accessing a cursor. The app should use a loader-based query to populate the list view for the main book activity, and a simple asynchronous resolver-based query (using `ISimpleQueryListener<T>`) to populate the view for the book detail activity. All insertion, deletion and update operations should also be asynchronous (using the asynchronous content resolver).

Note that your apps should never load data on the UI thread, and therefore they should never use the `startManagingCursor` or `managedQuery` operations, or the constructor for `SimpleCursorAdapter` that takes a cursor as its argument. However it is all right in general to use the `SimpleCursorAdapter` class, using the second constructor that provides a flag that indicates that the query should not be managed by the activity. Just do not use the first, deprecated constructor that causes queries to be managed on the UI thread. You are defining a custom adapter in this assignment because of the demands of the application, for custom rendering of cursor data in a list view.

Part 2: Persistent Chat App

In this second part of the assignment, you will extend the chat server app from the previous assignment. As with the first part of the assignment, you are required to replace the use of a SQLite database with a content provider. Define a single content provider with two tables, visible to the app, and distinguished by their URIs that have the same authority but different content paths. One table stores the messages that have been received, while the second table stores information about the peers from whom we have received messages. As with the bookstore app, you should define a contract and a provider class for this content provider, as well as entity classes for messages and peers, and a manager class, using the same package structure as outlined above for the bookstore app.

For this assignment, you should provide just one solution with two apps, `ChatClient` and `ChatServerWithContentProvider`. This should follow the guidelines for the second of the bookstore solutions above: Use cursor loaders with asynchronous query factories for querying all messages received so far, all peers from whom we have received messages, and all of the messages that we have received from a particular peer. Use a simple asynchronous query (based on an asynchronous content resolver) to retrieve the details for a particular peer with whom we have been in communication.

Add a `PreferenceActivity` class for defining the client name in the app. This activity can be launched from the action bar in the main app, accepts a client name as input, and remembers that name as a preference or shared preference. The name is prepended to every message that the client sends (at the front of the message). On the server, the name is separated from each incoming request.

Submitting Your Assignment

Once you have your code working, please follow these instructions for submitting your assignment:

1. Export your (four) Android Studio projects to the file system: book store with content provider, book store with entity manager, chat client and chat server.
2. Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory Humphrey_Bogart.
3. In that directory you should provide the Android Studio projects for your apps.
4. Also include in the directory a report of your submission. This report should be in PDF format. Do not provide a Word document.

In addition, record short flash, mpeg, avi or Quicktime videos of a demonstration of your assignment working. Make sure that your name appears at the beginning of the video. For example, put your name in the title of the app. *Do not provide private information such as your email or cwid in the video*. You can upload this video to the folder you are provided with in Google Drive. The video must be uploaded on time for the assignment as a whole to be considered as on time. The time of submission is based on the latest of the time you submitted your code and report to Canvas, and the time you uploaded your videos to Google Drive.

Alternatively, if you wish, you may use the Jing[1] app to record this video. This app will record a short video and provide you with a link to a Flash version of it in the cloud. Provide this link in your documentation. As with all cloud-based solutions, the problem with the Jing solution is your loss of privacy, so you are encouraged to investigate solutions that do not require use of the cloud. Be careful of any "free" apps that you download to do the recording, there are known cases of such apps containing Trojan horses, including key loggers.

Your solution should be uploaded via the Canvas classroom. Your solution should consist of a zip archive with one folder, identified by your name. Within that folder, you should have four Android Studio projects, for the apps you have built. You should also provide a report in the root folder, called README.pdf, that contains a report on your solution, as well as videos demonstrating the working of your assignments (unless the videos were uploaded to Google Drive instead).

[1] http://www.techsmith.com/Jing