CS 522—Spring 2018
Mobile Systems and Applications
Assignment Five—Content Providers and Entity Managers

Set your minimum SDK version to Lollipop (Android 5.1, API 22) for your submission for this assignment. Your app theme should extend `android:Theme.Material.Light.DarkActionBar`We will be testing your app on a virtual Google Nexus 5X running API 22, so you should ensure that your app works on that platform, although you may develop on another device, e.g., your personal telephone.

Part 1: Book Store

In the previous assignment, you implemented a book store app that stored the contents of the shopping cart in a content provider. In this assignment, you will provide a similar app, but using the abstractions discussed in class for hiding the complexity of background thread management, while providing a type-safe API for the presentation layer. As before, the 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 before, define a custom cursor adapter that extracts the first author name from the list of book authors and just displays that.

As before, 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.

Also as before, define an entity class, `Book`, for book entities stored in the database. 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.

Call your solution for this assignment `BookstoreEntityManager`. This follows the two guidelines for the previous solution repeated above: define a contract, provider and entity class. However, for this solution, **all** content provider operations should be asynchronous

(not just the main query). 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. All insertion, deletion and update operations should also be asynchronous (using the asynchronous content resolver).

Note that your apps 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 (your specialization of) 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 previous 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 `ChatServerEntityManager`. This should follow the guidelines for the bookstore solution 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.

Submitting Your Assignment

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

1. You should submit three Android Studio projects: book store with entity manager, chat client, and chat server with entity manager.
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. You should also provide APK files for your compiled projects.
5. Also include in the directory a report of your submission. This report should be in PDF format. Do not provide a Word document. This report should summarize what you've done, with an emphasis on any assumptions you made about points not covered in the assignment specification. It should also identify where the projects and APK files are located in your submission.

In addition, record short mpeg or Quicktime videos of demonstrations of your assignment working. Make sure that your name appears at the beginning of each 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*. 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 videos demonstrating the working of your assignments.