CS 522—Fall 2020
Mobile Systems and Applications
Assignment Five—Content Providers and Entity Managers

In the previous assignment, you implemented a chat server app that stored chat messages sent to it 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 store the data using a SQLite database with two tables, identified by the content paths, `messages` and `peers`. You should again have activities for viewing a list of all messages, viewing a list of all chat peers (accessed via a menu), and viewing the details for a single chat peer, including a list of all their messages. The last activity is started when the user selects a chat peer from the list of peers in the second activity.

As before, define contracts, `MessageContract` and `PeerContract`, for the content provider, `ChatProvider`. Place the contracts classes in the `contracts` subpackage of your app, and the provider 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 entity classes, `Message`, for messages stored in the database, and `Peer`, for chat peers stored in the database. These should define entity fields, implementations of the `Parcelable` interface, constructor sfor initializing an entity from a cursor, and an operation `writeToProvider` for initializing a `ContentValues` object (for insertion into a provider) with the fields of the entity.

Unlike the previous assignment, **all** content provider operations should be asynchronous(not just the queries that populate list views). Furthermore, all access to the provider by the application should be defined through manager objects, `MessageManager` and `PeerManager`, that extend a generic abstract class `Manager<T>` as defined in the lecture materials (instantiating it with the `Message` and `Peer` entity types, respectively). The manager base class should define both synchronous and asynchronous content resolvers, and should be defined in the `managers` subpackage with the other manager classes. You will have to define the latter class, `AsyncContentResolver`, inheriting from `AsyncQueryHandler`. The manager base class should define generic methods for asynchronous queries (both using loaders and using the asynchronous content resolver), using factory methods defined in the `QueryBuilder` and `SimpleQueryBuilder` classes. The `MessageManager` and PeerManager classes 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 loader-based queries to populate the list view for the activities, but rather than using the loader manager and cursor loaders directly, instead use the

`QueryBuilder` class (through the query operations defined in the manager classes). All insertion, deletion and update operations should also be asynchronous (using the asynchronous content resolver). When a message is received from a peer, that message and peer should be upserted into the database using an asynchronous content resolver (which is only accessed through message and peer manager objects that provide type-safe asynchronous APIs for accessing the content provider). You will need to upsert the peer object first all, since you will need its primary key as a foreign key reference in the manager object, and then insert the message object:

```
peerManager.persistAsync(peer,
    id -> {
        message.senderId = id;
        messageManager.persistAsync(message);
    });
```

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 (a 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.

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: 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 completed rubric for your assignment.

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 two Android Studio projects, for the apps you have built. You should also provide videos demonstrating the working of your assignments. **Make sure that your video shows the server app being launched from the Android Studio project based on content providers and entity managers.**