Practical 14: Deleting data from a Room database

Introduction

This codelab (practical) follows on from 13 Part A: Room, LiveData, and ViewModel. This codelab gives you more practice at using the API provided by the Room library to implement database functionality. You will add the ability to delete specific items from the database.

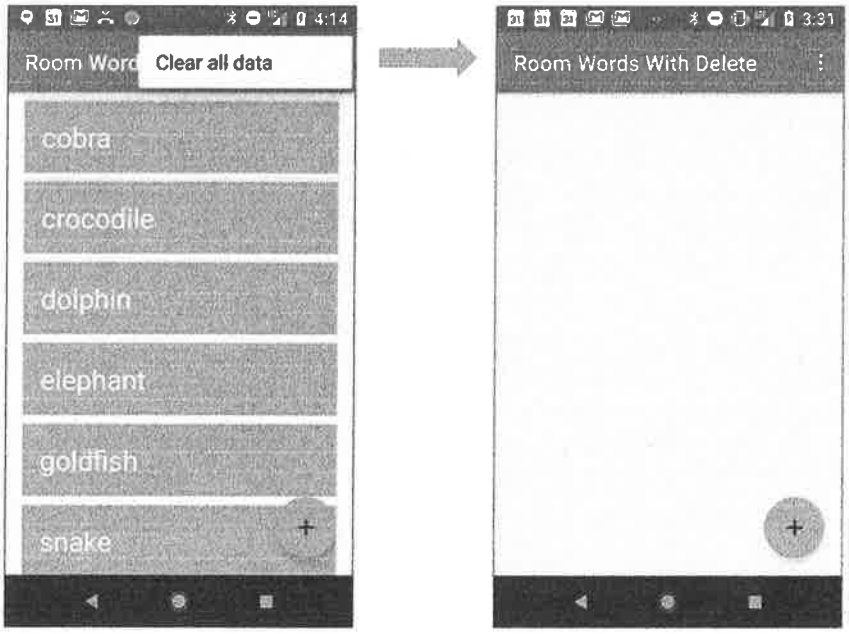
- Update the RoomWordsSample app to keep data when the app closes.
- Allow users to delete all words by selecting an **Options** menu item.
- Allow users to delete a specific word by swiping an item in the list.
- Optionally, in a coding challenge, extend the app to allow the user to update existing words.

App overview

You will extend the RoomWordsSample app that you created in the previous codelab. So far, that app displays a list of words, and users can add words. When the app closes and re-opens, the app re-initializes the database. Words that the user has added are lost.

In this practical, you extend the app so that it only initializes the data in the database if there is no existing data.

Then you add a menu item that allows the user to delete all the data.



You also enable the user to swipe a word to delete it from the database.

Task 1: Initialize data only if the database is empty

The RoomWordsSample app that you created in the previous practical deletes and re-creates the data whenever the user opens the app. This behavior isn't ideal, because users will want their added words to remain in the database when the app is closed. (Solution code for the previous practical is in GitHub.)

In this task you update the app so that when it opens, the initial data set is only added if the database has no data.

To detect whether the database contains data already, you can run a query to get one data item. If the query returns nothing, then the database is empty.

Note: In a production app, you might want to allow users to delete all data without re-initializing the data when the app restarts. But for testing purposes, it's useful to be able to delete all data, then re-initialize the data when the app starts.

1.1 Add a method to the DAO to get a single word

Currently, the **WordDao** interface has a method for getting all the words, but not for getting any single word. The method to get a single word does not need to return **LiveData**, because your app will call the method explicitly when needed.

1. In the **WordDao** interface, add a method to get any word:

```
@Query("SELECT * from word_table LIMIT 1")
  Word[] getAnyWord();
```

Room issues the database query when the **getAnyWord()** method is called and returns an array containing one word. You don't need to write any additional code to implement it.

1.2 Update the initialization method to check whether data exists

Use the **getAnyWord()** method in the method that initializes the database. If there is any data, leave the data as it is. If there is no data, add the initial data set.

1. **PopulateDBAsync** is an inner class in **WordRoomDatbase.** In **PopulateDBAsync,** update the **doInBackground()** method to check whether the database has any words before initializing the data:

```
@Override
protected Void doInBackground(final Void... params) {

    // If we have no words, then create the initial list of words
    if (mDao.getAnyWord().length < 1) {
        for (int i = 0; i <= words.length - 1; i++) {
            Word word = new Word(words[i]);
            mDao.insert(word);
        }
    }
  return null;
}
```

2. Run your app and add several new words. Close the app and restart it. You should see the new words that you added, as the words should now persist when the app is closed and opened again.

Task 2: Delete all words

In the previous practical, you used the **deleteAll()** method to clear out all the data when the database opened. The **deleteAll()** method was only invoked from the **PopulateDbAsync** class when the app started. You will now make the **deleteAll()** method available through the **ViewModel** so that your app can call the method whenever it's needed.

Here are the general steps for implementing a method to use the **Room** library to interact with the database:

- Add the method to the DAO, and annotate it with the relevant database operation. For the **deleteAll()** method, you already did this step in the previous practical.
- Add the method to the **WordRepository** class. Write the code to run the method in the background.
- To call the method in the **WordRepository** class, add the method to the **WordViewModel**. The rest of the app can then access the method through the **WordViewModel**.

2.1 Add deleteAll() to the WordDao interface and annotate it

1. In **WordDao**, check that the **deleteAll()** method is defined and annotated with the SQL that runs when the method executes:

```
@Query("DELETE FROM word_table")
void deleteAll();
```

2.2 Add deleteAll() to the WordRepository class

Add the **deleteAll()** method to the **WordRepository** and implement an **AsyncTask** to delete all words in the background.

1. In **WordRepository**, define **deleteAllWordsAsyncTask** as an inner class. Implement **doInBackground()** to delete all the words by calling **deleteAll()** on the DAO:

```
private static class deleteAllWordsAsyncTask extends AsyncTask<Void, Void, Void> {
  private WordDao mAsyncTaskDao;

  deleteAllWordsAsyncTask(WordDao dao) {
    mAsyncTaskDao = dao;
  }
```

```
    @Override
    protected Void doInBackground(Void... voids) {
        mAsyncTaskDao.deleteAll();
        return null;
    }
}
```

2. In the **WordRepository** class, add the **deleteAll()** method to invoke the **AsyncTask** that you defined.

```
public void deleteAll()  {
    new deleteAllWordsAsyncTask(mWordDao).execute();
}
```

2.3 Add deleteAll() to the WordViewModel class

Make the **deleteAll()** method available to the **MainActivity** by adding it to the **WordViewModel**.

1. In the **WordViewModel** class, add the **deleteAll()** method:

```
public void deleteAll() {mRepository.deleteAll();}
```

Task 3: Add an Options menu item to delete all data

Next, you add a menu item to enable users to invoke **deleteAll()**.

Note: The production version of your app must provide safeguards so that users do not accidentally wipe out their entire database. However, while you develop your app, it's helpful to be able to clear out test data quickly. This is especially true now that your app does not clear out the data when the app opens.

3.1 Add the Clear all data menu option

1. In **menu_main.xml**, change the menu option **title** and **id**, as follows:

```
<item
  android:id="@+id/clear_data"
  android:orderInCategory="100"
  android:title="@string/clear_all_data"
  app:showAsAction="never" />
```

2. In **MainActivity**, implement the **onOptionsItemSelected()** method to invoke the **deleteAll()** method on the **WordViewModel** object.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  int id = item.getItemId();

  if (id == R.id.clear_data) {
    // Add a toast just for confirmation
    Toast.makeText(this, "Clearing the data...",
        Toast.LENGTH_SHORT).show();

    // Delete the existing data
    mWordViewModel.deleteAll();
    return true;
  }

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

3. Run your app. In the **Options** menu, select **Clear all data**. All words should disappear.
4. Restart the app. (Restart it from your device or the emulator; don't run it again from Android Studio) You should see the initial set of words.

   Note: After you clear the data, re-deploying the app from Android Studio shows the initial data set again. Opening the app shows the empty data set.

Task 4: Delete a single word

Your app lets users add words and delete all words. In Tasks 4 and 5, you extend the app so that users can delete a word by swiping the item in the **RecyclerView**.

Again, here are the general steps to implement a method to use the **Room** library to interact with the database:

- Add the method to the DAO, and annotate it with the relevant database operation.
- Add the method to the **WordRepository** class. Write the code to run the method in the background.
- To call the method in the **WordRepository** class, add the method to the **WordViewModel**. The rest of the app can then access the method through the **WordViewModel**.

4.1 Add deleteWord() to the DAO and annotate it

1. In **WordDao**, add the **deleteWord()** method:

```
@Delete
void deleteWord(Word word);
```

Because this operation deletes a single row, the **@Delete** annotation is all that is needed to delete the word from the database.

4.2 Add deleteWord() to the WordRepository class

1. In **WordRepository**, define another **AsyncTask** called **deleteWordAsyncTask** as an inner class. Implement **doInBackground()** to delete a word by calling **deleteWord()** on the DAO:

```
private static class deleteWordAsyncTask extends AsyncTask<Word, Void, Void> {
  private WordDao mAsyncTaskDao;

  deleteWordAsyncTask(WordDao dao) {
    mAsyncTaskDao = dao;
  }
```

```
    @Override
    protected Void doInBackground(final Word... params) {
        mAsyncTaskDao.deleteWord(params[0]);
        return null;
    }
}
```

2. In **WordRepository**, add the **deleteWord()** method to invoke the **AsyncTask** you defined.

```
public void deleteWord(Word word)  {
    new deleteWordAsyncTask(mWordDao).execute(word);
}
```

4.3 Add deleteWord() to the WordViewModel class

To make the **deleteWord()** method available to other classes in the app, in particular, **MainActivity**, add it to **WordViewModel**.

1. In **WordViewModel**, add the **deleteWord()** method:

```
public void deleteWord(Word word) {mRepository.deleteWord(word);}
```

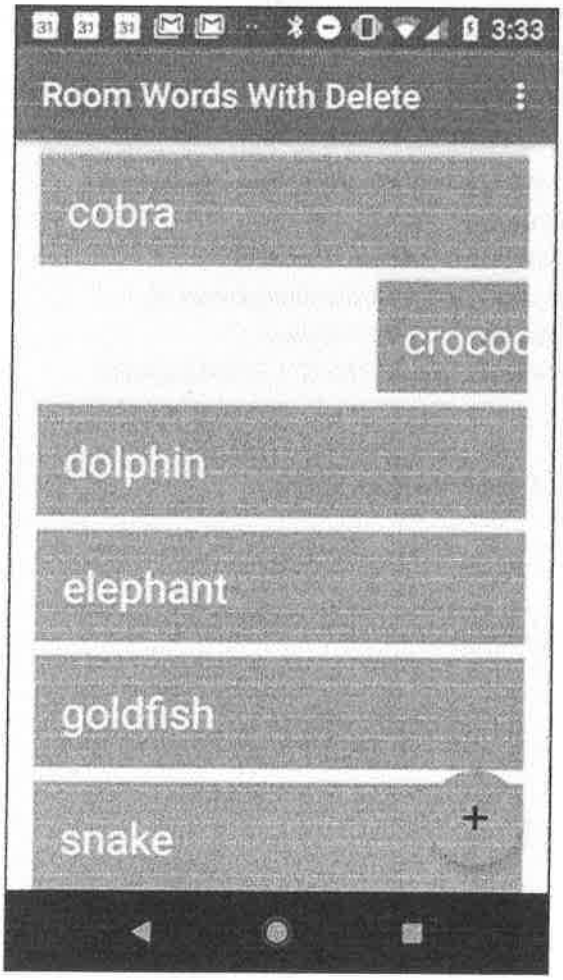
You have now implemented the logic to delete a word. As yet, there is no way to invoke the delete-word operation from the app's UI. You fix that next.

Task 5: Enable users to swipe words away

In this task, you add functionality to allow users to swipe an item in the RecyclerView to delete it.

Use the ItemTouchHelper class provided by the Android Support Library (version 7 and higher) to implement swipe functionality in your RecyclerView. The ItemTouchHelper class has the following methods:

- onMove() is called when the user moves the item. You will not implement any move functionality in this app.
- onSwipe() is called when the user swipes the item. You implement this method to delete the word that was swiped.



5.1 Enable the adapter to detect the swiped word

1. In WordListAdapter, add a method to get the word at a given position.

```
public Word getWordAtPosition (int position) {
  return mWords.get(position);
}
```

2. In MainActivity, in onCreate(), create the ItemTouchHelper. Attach the ItemTouchHelper to the RecyclerView.

```
// Add the functionality to swipe items in the
// recycler view to delete that item
ItemTouchHelper helper = new ItemTouchHelper(
    new ItemTouchHelper.SimpleCallback(0,
        ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT) {
          @Override
          public boolean onMove(RecyclerView recyclerView,
                                RecyclerView.ViewHolder viewHolder,
                                RecyclerView.ViewHolder target) {
            return false;
          }

          @Override
          public void onSwiped(RecyclerView.ViewHolder viewHolder,
                               int direction) {
            int position = viewHolder.getAdapterPosition();
            Word myWord = adapter.getWordAtPosition(position);
            Toast.makeText(MainActivity.this, "Deleting " +
                  myWord.getWord(), Toast.LENGTH_LONG).show();

            // Delete the word
            mWordViewModel.deleteWord(myWord);
          }
        });

helper.attachToRecyclerView(recyclerView);
```

Things to notice in the code:

onSwiped() gets the position of the ViewHolder that was swiped:

```
int position = viewHolder.getAdapterPosition();
```

Given the position, you can get the word displayed by the ViewHolder by calling the getWordAtPosition() method that you defined in the adapter:

```
Word myWord = adapter.getWordAtPosition(position);
```

Delete the word by calling deleteWord() on the WordViewModel:

```
mWordViewModel.deleteWord(myWord);
```

Now run your app and delete some words

1. Run your app. You should be able to delete words by swiping them left or right.