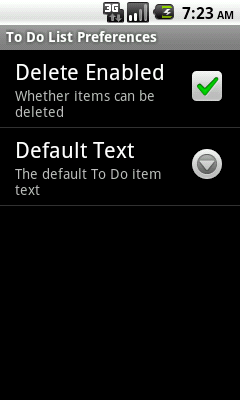
LAB: Module 4 – Storing the TODO List

In this lab, you will continue the development of the TODO List Manager from Module 3. This time, you will add to the TODO List Manager persistence capabilities, so that the list of TODO items is not lost whenever the application is closed.

# Part 1

1. You can continue working on your TODO List Manager from Module 3, or start from the instructor-supplied solution (in the **Solutions\ToDoListManager** directory). Either way, make a backup of your code before modifying it.

TIP: To import an existing project into Eclipse, select File 🡪 Import 🡪 General 🡪 Existing Projects into Workspace, choose the appropriate directory and click Refresh to let Eclipse locate the project for you.

1. Add a Preferences screen to the application, with two preferences: “Delete Enabled” and “Default Text”. The first controls whether items can be deleted (from the options menu or the context menu), and the second controls the default text in the new item text field.
   1. Create a new activity, ToDoPreferencesActivity, derived from PreferenceActivity.

TIP: Don’t forget to add the new activity to your application manifest.

* 1. Create a new menu item in the main activity that invokes the new preferences activity.
  2. To monitor preference changes, you can use the registerOnSharedPreferenceChangeListener() of the SharedPreferences class.

# Part 2

1. Whenever a user adds, removes, or edits a TODO item, save the list of TODO items in a file. You can use Java’s object serialization facilities, such as ObjectOutputStream, or devise your own file format.
2. In the main activity’s onCreate() method, load the list of TODO items from the file.

# Part 3

1. Remove the code added in Part 2, or start from the instructor-supplied solution. (See Step 1.)
2. Design a database schema to store the TODO items. If possible, use just one table – it will make the rest of the lab much easier. Remember that the overarching goal is to store the TODO item information in the database, and perform all management operations on the persistent store.
3. Create a ToDoListDatabaseHelper class that derives from SQLiteOpenHelper.
   1. In its onCreate() method, issue an execSQL() call to create the database table.
   2. Don’t forget to create an automatically-generated ID column named \_id (this is just a convention, but it’s used throughout any Android database or application).
   3. Create public constants that will make it easier to access your database. For example, create a constant for the table name, the column names, and the entire column set.

TIP: Remember that you can always examine the current database state using adb shell and the sqlite3 tool, as explained in the module slides.

1. Create a ToDoDAL class that wraps the useful operations on TODO items that you will need to perform from the application. You need to implement only the operations you need; the following is a possible list of what you might need:
   1. Insert a new item – take a ToDoItem and insert it into the database.
   2. Update an item – take a ToDoItem and update its database values.
   3. Delete an item – given an ID or a ToDoItem, delete it from the database.
   4. Retrieve all TODO items – returns a Cursor to all the TODO items.

TIP: You might want to implement also another method that returns a collection of TODO items, or at least a method that converts the current cursor position to a TODO item instance. Note that if you have more than one class representing TODO items, this method will have to know which class to instantiate, based on the information in the database table.

* 1. Retrieve a specific TODO item given its ID.

1. Modify your adapter derived from ArrayAdapter<T> to derive from SimpleCursorAdapter instead.
   1. Your overriding getView() method will have to obtain the information to place in the view from the cursor, using getCursor() and moveToPosition().
2. Modify the main activity to use your new adapter. You will need to create an instance of the ToDoDAL and retrieve from it the cursor used for UI binding.
   1. When adding a new item, use the DAL to insert it to the database.
   2. When removing an item, use the DAL to remove it.
   3. When updating an item, use the DAL to update it.

TIP: You might wonder how the update and remove item UI handlers can find the TODO item object to work on. Previously, the list view’s position was in sync with the list of items used for the data binding; now the list of items is not in sync because you use the DAL to make the updates. One way to solve the problem is to maintain a list of items beside the cursor, and keep it up to date by syncing it with the cursor whenever you insert a new item, remove an item, or update an item.

For example, if items is a list of items and cursor is the cursor obtained from the DAL, then you can use code similar to the following to “sync” the list of items with the cursor:  
  
**private** **void** updateItemsArray() {  
 items.clear();  
 cursor.requery();  
 **if** (cursor.moveToFirst()) {  
 **do** {  
 items.add(ToDoDAL.*fromCurrentCursorPosition*(cursor));  
 } **while** (cursor.moveToNext());  
 }  
}

1. Next, you should make sure that the application’s UI state is maintained even if the system decides to kill it.
   1. In the main activity you should persist the user’s text input into the “new task” field, and whether the text field is visible or not.
   2. Use the onSaveInstanceState() and onRestoreInstanceState() lifecycle events to store this information and retrieve it later.

# Part 4

1. Implement a content provider that exposes TODO item information from your existing database. Test your content provider from another application. (You can use the **DemoCode\WorkingWithData** folder as an example of a full-featured content provider.)