Stanford University, CS 193A Homework 2: To-Do List (Pairs allowed)

Assignment idea and spec by Marty Stepp.

This assignment practices material from this week, including widgets, file input, and saving/loading app state.

This is a **pair assignment**. You may complete it alone or with a partner. Please read our Pairs web page for some of our policies and suggestions if you do plan to work in a pair.

File and Project Names:

- Your **Android Studio project**'s name must be EXACTLY the following, case-sensitive, including the underscores:
    - `cs193a_hw2_`***`SUNETID`*** (example: `cs193a_hw2_ksmith12`)

  Or if you are working in a pair, please list both partners' SUNetIDs separated by an underscore:
    - `cs193a_hw2_`***`SUNETID1_SUNETID2`*** (example: `cs193a_hw2_ksmith12_tjones5`)
- Your activity's name must be `ToDoListActivity`, case sensitive, and it must be stored in a file named **ToDoListActivity.kt**. You may add other classes and files to your app if you like, but you must have this exact activity file name at a minimum, spelled and capitalized exactly as it is written here.
- When you are finished, ZIP your project (instructions) into a file with the following exact file name, including spelling and capitalization:
    - `cs193a_hw2_`***`SUNETID`***`.zip` (example: `cs193a_hw2_ksmith12.zip`)
    - or `cs193a_hw2_`***`SUNETID1_SUNETID2`***`.zip` if you work in a pair (example: `cs193a_hw2_ksmith12_tjones5.zip`)
- Turn in link: Turn in HW2 here. (You only need to turn in one copy if you work in a pair.)

Program Description and Functionality Requirements:

Write a simple **to-do list** app that has a list of tasks that the user needs to complete. (We strongly suggest using a `ListView` to represent the to-do list.) Initially the app is empty and has nothing in the to-do list.

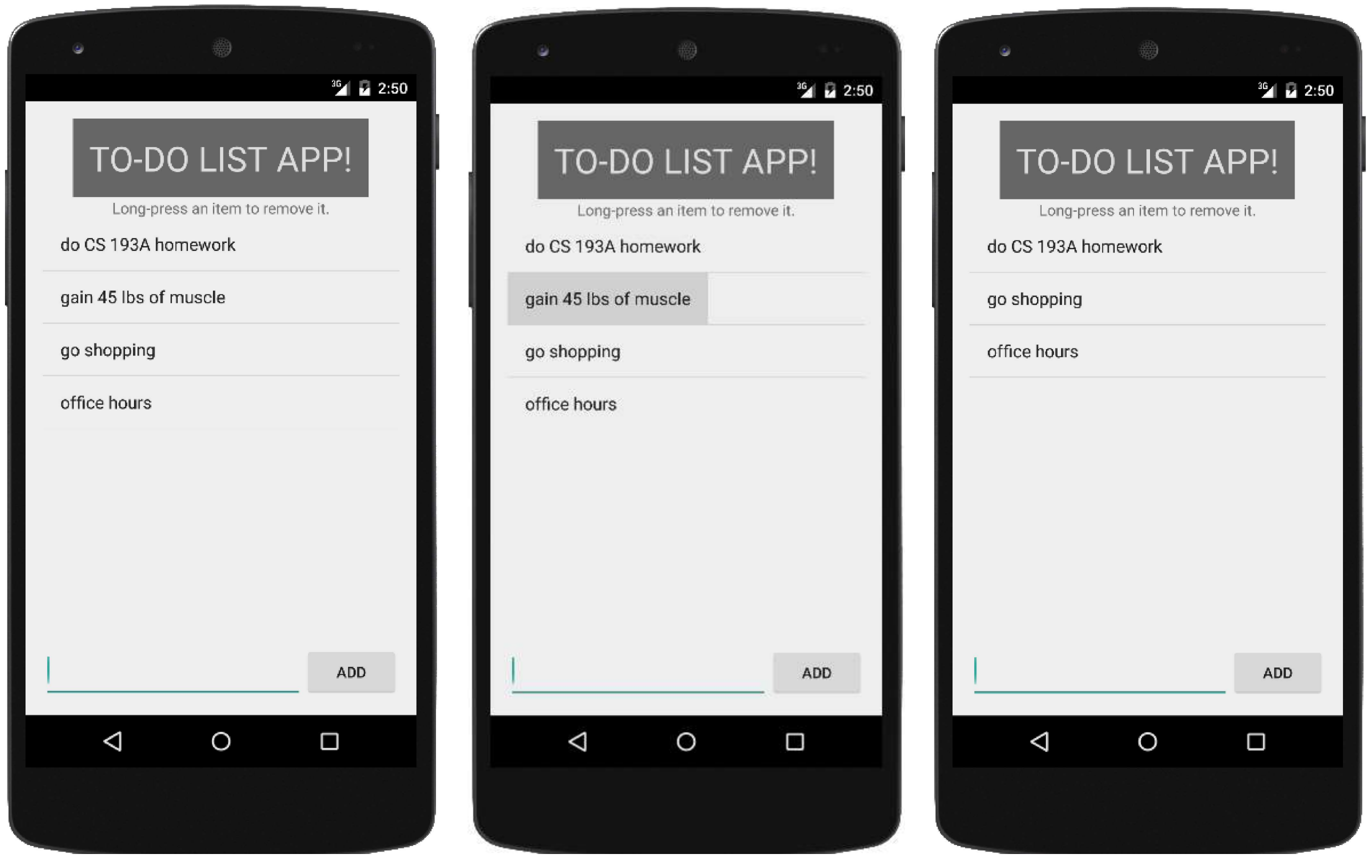

Figure: User long-clicking on second list item to delete it

Here are the minimum required features your app must have:

1. **ADDING ITEMS:** Provide a way for the user to type in items into their to-do list. For example, you could provide an `EditText` at the bottom of the screen such that when the user types text into it and clicks an Add button, the new item will be added to the top or bottom of the list. It's up to you whether new items should be added to the top or the bottom of the list.
2. **VIEWING ITEMS:** This is sort of obvious, but the items the user has added should be visible as a list (such as a `ListView`). When an item is added or removed, the list should update itself immediately.
3. **REMOVING ITEMS:** Provide a way to remove an item from the list. You could achieve this by attaching an event listener that removes a list item when that item is clicked by the user. Or if you want to try something slightly different, try making it remove an item when the user performs a "long click" (pressing and holding the mouse on an item). You can do this by calling the `setOnItemLongClickListener` method of your list and passing an anonymous `AdapterView.OnItemLongClickListener` class. (Android Studio can help you auto-generate the skeletons of these anonymous listener classes if you press Ctrl-Space in the editor at the right place in the code.)

4. **SCROLLING:** If the to-do list becomes very long, it may be too large to fit on the screen. Make sure that it is possible for the user to swipe vertically to scroll through the list. (You might or might not need to actually write extra code to handle this; a `ListView` can handle its own scrolling if sized and laid out properly. We just want to state that your app must have this functionality.)
5. **MOVE TO TOP:** Provide some way to move a given item in the to-do list to the top of the list. For example, you could make it so that clicking on an item, or long-clicking on it, moves it to the top. Or you could provide a button that, when clicked, moves the most recently added item from the bottom of the list to the top.
6. **ONE ADDITIONAL FEATURE:** You should add at least one other feature to your app of your own choosing. The feature can be any of the "extra" features listed below, for example, or a different feature that you come up with on your own. (This is not meant to be a huge amount of extra work, so if you want to pick a modest feature that is not very many lines of code to implement, that is fine. Please post on the class discussion board if you have questions about whether a given feature is suitable here.)

Note: If the items in your to-do list are stored into an `ArrayList`, by default the app's GUI won't notice when you add or remove an item from the `ArrayList` variable. That is, you'll modify the `ArrayList`'s state in your code, but the graphical list on the screen won't update to match. To fix this, you have to call the method `notifyDataSetChanged()` on your list's `ArrayAdapter` to tell it that the underlying array list has changed. To be able to do this, of course, you'll have to save your `ArrayList` and your `ArrayAdapter` as private fields inside your activity.

Other miscellaneous functionality requirements that you must follow in this program:

- Your app does not need to match the exact appearance of the screenshots in this document. But you should use a reasonable **layout** so that the various widgets are visible and can be interacted with on a standard AVD phone device such as a Nexus 5X.
- Your submission must be **your own work** and not written by someone else.
- Your submission must be **new work** for CS 193A and cannot be a project you submitted for another class, such as CS 108 or CS 147.

Extra Features:

If you are interested in adding more functionality to your app beyond what is required, here are some options. You may also use any of these features as your "one additional feature" as described above.

- **Custom list item layout:** Rather than the default list item appearance, use a custom layout XML to modify the list item appearance to your liking. You could change the font style, add an icon to each list item, etc.

- **Prioritized list:** Provide a way for the user to indicate the priority or urgency of each item in the to-do list. When new items are added to the list, it should sort in descending order of priority so that the most urgent items are shown first in the list. You could also implement a somewhat simpler version of this where the user can "star" an item that is urgent.
- **Separate "completed" items:** When the user has completed an item from the list, instead of just removing them permanently from the screen, indicate that the item is "completed" by changing its appearance, putting the completed item somewhere else on the screen (a second list?), etc. so that the user can track all of the items he/she completed over time.
- **Items with due dates:** Make it possible for the user to specify a date when each to-do item must be completed. You may want to sort the list's items by required completion date. You may want to look up `DatePicker` or `DatePickerDialog` to help accomplish this.
- **Persistent (saved) list:** To make a more robust app, you should also consider **saving the list to a file** when the to-do list is modified. You can save the to-do list to the device's storage so that it will survive a reboot. To do this, you'd need to write code that reads/writes files to the system's internal storage:

```
// 1) write output to a file
val out = PrintStream(openFileOutput("filename.txt", MODE_PRIVATE))
out.println(text)
out.close()
```

```
// 2) read input from a file
val scan = Scanner(openFileInput("filename.txt"))
while (scan.hasNextLine()) {
    val line = scan.nextLine()
    do something with the line
}
```

- **Sound effects:** Make your app play a sound effect when an item is added, removed, or rearranged from the list. We have not learned about playing sounds yet, but we will do so next week.

Style Requirements:

This course will generally have more lenient coding style requirements than a class like CS 106A/B. But we do have some important style requirements below that we ask you to follow. If you do not follow all of these requirements, you will not receive full credit for the assignment.

- **Comments:** Write a **comment header** in your main activity **.kt** file containing your name and email address along with the name of your app and a very brief description of your program, along with any special instructions that the user might need to know in order to use it properly (if there are any). Also write a brief comment header at the top of every **method** in your **.kt** code that explains the method's purpose. All of these comments can be brief (1-2 lines or sentences); just explain what the method does and/or when/why it is called. You do not need to use any particular comment format, nor do you need to document exactly what each parameter or return value does. The following is a reasonable example of a comment header at the top of a **.kt** activity file:

```
/*
 * Kelly Smith <ksmith12@stanford.edu>
 * CS 193A, Winter 2049 (instructor: Mrs. Krabappel)
 * Homework Assignment 2
 * NumberGame 2.05 - This app shows two numbers on the screen and asks
 * the user to pick the larger number.  Perfect for Berkeley students!
 * Note: Runs best on big Android tablets because of 1000dp font choice.
 */
```

- **Redundancy:** If you perform a significant operation that is exactly or almost exactly the same in multiple places in your code, avoid this redundancy, such as by moving that operation into a helping method. See the lecture code from our lecture #1 for examples of this.
- **Reduce unnecessary use of global variables and private fields:** While you are allowed to have private fields ("instance variables") in your program, you should try to minimize fields unless the value of that field is used in several places in the code and is important throughout the lifetime of the app.
- **Reduce mutability:** Use `val` rather than `var` as much as possible when declaring Kotlin variables. Use immutable collections instead of mutable ones where appropriate.
- **Naming:** Give variables, methods, classes, etc. descriptive names. For example, other than a `for` loop counter variable such as `int i`, do not give a variable a one-letter name. Similarly, if you give an `id` property value to a widget such as a `TextView`, apply a similar style standard as if it were a variable name and choose a descriptive name.

Survey: After you turn in the assignment, we would love for you to fill out our optional anonymous CS 193A homework survey to tell us how much you liked / disliked the assignment, how challenging you found it, how long it took you, etc. This information helps us improve future assignments.

Honor Code Reminder: Please remember to follow the **Honor Code** when working on this assignment. Submit your own work and do not look at others' solutions. Also please do not give out your solution and do not place a solution to this assignment on a public web site or forum. If you need help, please seek out our available resources to help you.