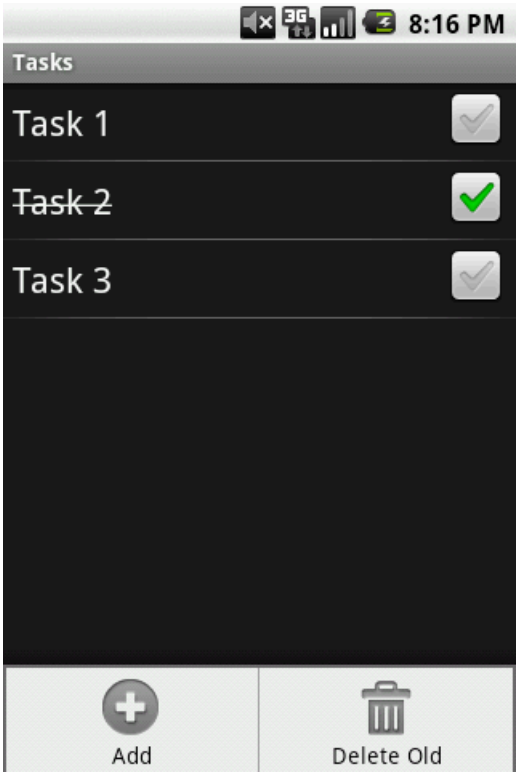
CS 2046 - Assignment 1	**Due**: Monday, 11/1, 11:59pm on CMS

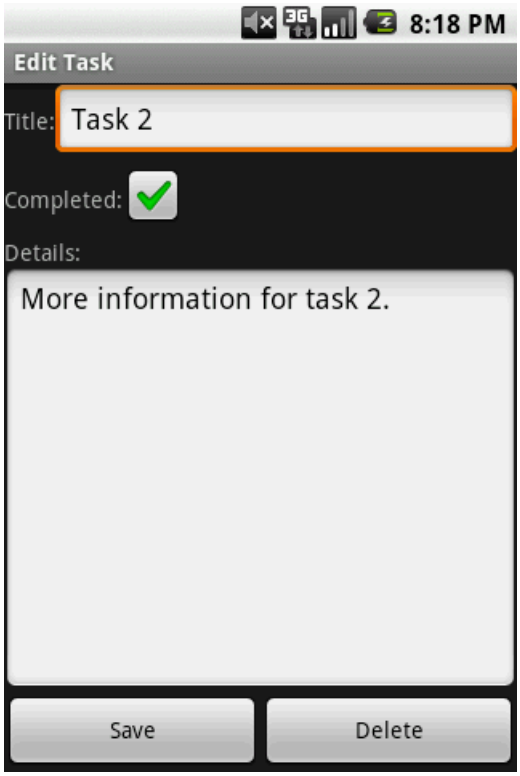
Building a UI for Tasks

Your assignment is to design the user interface for Tasks, a to-do list application. In future assignments, you will link the UI to a database to make the application functional, and synchronize the to-do list with an online service. For this assignment, we will use some dummy data, and most of the UI elements will do nothing.

Tasks consists of two Activities: TaskList, which displays the list of tasks, and TaskEdit, which allows the user to edit the different attributes a task can have. As you can see in the source code for the Task class, which is an inner class of TaskList, a task consists of a title, details, and a boolean stating whether or not the task has been completed.

The finished Activities will look like this:





On CMS, you can download an Eclipse project containing the skeleton code for this assignment. You can open the project in Eclipse by going to File → Import..., selecting General → Existing Projects into Workspace, and then selecting the skeleton code archive. This will copy the code into your Eclipse workspace. You will be writing your project for Android 1.5 - if you do not have this version of the SDK installed, make sure you install it by following the instructions on the course website.

Task Breakdown (no pun intended)

It is suggested that you complete the assignment in this order:

- *Design the Activity UIs.* You will design the XML layouts for both TaskList and TaskEdit. The layouts are located in `res/layout/task_list.xml` and `res/layout/task_edit.xml`, respectively.

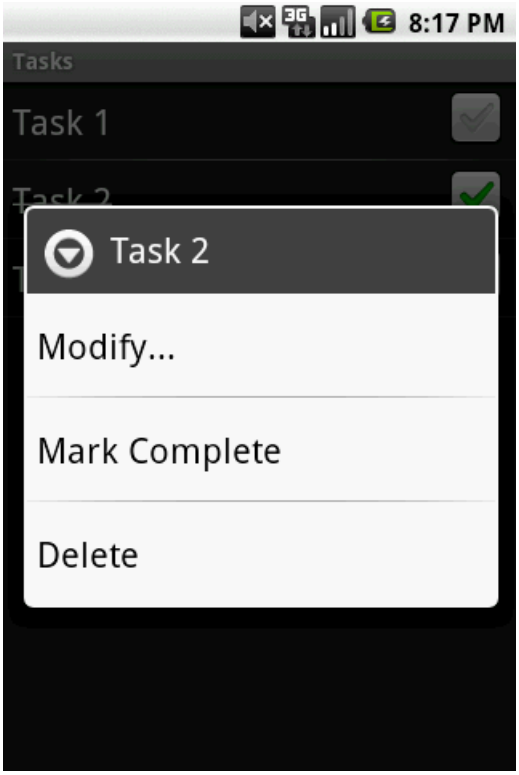
  TaskList's layout is fairly simple, and most of it is provided for you. The only addition you will need to make is an empty view, which will appear when the list is empty. This view should have id `no_tasks` and display whatever text is located in `res/values/strings.xml` under `no_tasks`.

  On the other hand, TaskEdit's layout file is essentially empty, so you will need to define this layout from scratch. You should model your layout after the above screenshot. As discussed in class, there is always more than one way to create a layout, and so there is no one correct answer to this part. As long as your interface looks about the same as the screenshot, you will receive full credit for this section. Remember, if you get stuck, a good resource is the Hello, Views tutorial.

  Note that the menu in the screenshot above is not part of the XML layout, but is defined programatically to appear when the Menu button is pressed.

- *Complete onCreate in TaskList.* You have been supplied with code that initializes mTasks, the class variable referring to the task ListView, and which sets the choice mode of the ListView. The choice mode attribute specifies that multiple items in the list can be selected (i.e. marked complete) - for other applications of ListViews, you can set this attribute so that only one item can be selected, or (the default) that no items can be selected. Note that this corresponds to the UI XML attribute `android:choiceMode`; however, because of a bug in Android 1.5, the choice mode is not set when this attribute is used.

  You will need to use the `ListView.setEmptyView` method to set the empty view to the view you defined in the layout XML. The other two steps listed in the TODO for `onCreate` are taken care of below.



- *Create the context menu.* ListViews support displaying context menus. When you click and hold an item in the ListView, the Activity can display a context menu for that item. You will create the above menu. There are three steps to enable your Activity to show context menus:

  - Implement `onCreateContextMenu`. This method is called when you long-press an item and the associated menu must be created. We have supplied the code which finds the corresponding Task for the row that was long-pressed; you should set the title of the context menu to the title of the task using `menu.setHeaderTitle`, and then use `menu.add` to add entries for the three commands above. `ContextMenu.add` takes four arguments:

    * groupId: Items in a menu can be grouped together. This isn't being used here, so you should pass `Menu.NONE`.
    * itemId: A unique identifier for the item, so we know which item was clicked in `onContextItemSelected`. These are provided for you as static constants starting with `CONTEXT_MENU_ITEM_`.
    * order: Where this item should appear in the menu. Items with larger numbers appear below items with smaller numbers.
    * titleRes: The resource for the string which should be displayed. These are defined for you as `modify_task`, `mark_complete_task`, and `delete_task` in `strings.xml`.

  - Implement `onContextItemSelected`. Most of the infrastructure for this method is given for you; for this assignment, the Mark Complete and Delete commands will do nothing, while the Modify... command should launch the TaskEdit activity. The Intent that you pass to startActivity should contain extras which define the selected task - the title, isCompleted, and details variables. See `Intent.putExtra` for how to do this. You can name these extras whatever you like; these names will be used to recover these values in the TaskEdit activity. The selected task is available as `taskArray[(int) info.id]`.
  - Set the Activity as the handler for context menu requests. By default, the ListView does not know which class will handle the above two functions, and so context menus are disabled. In `onCreate`, you should use `ListView.setOnCreateContextMenuListener` to set this Activity instance as this handler.

- *Create the options menu.* In addition to the context menu for each task, you can create a menu which applies to the entire Activity. You will do this by implementing `onCreateOptionsMenu` in TaskList. Once again, you need to call `menu.add` to add the two menu items, Add and Delete Old. Note that `menu.add` returns the MenuItem that was added; you should call `MenuItem.setIcon` to set the icons for these menu items. Android provides built-in icons in the `android.R.drawable` class; you may use `android.R.drawable.ic_menu_add` as the Add icon and `android.R.drawable.ic_menu_delete` as the Delete Old icon. You should also use the `add_task` and `delete_old_tasks` strings.

  Finally, be sure that you read the documentation for `Activity.onCreateOptionsMenu` carefully. This is where the implementation contract that you must satisfy is specified, including whether or not you should call the superclass method, and what you should return. If you find that your options menu isn't appearing, it is likely because you have violated this contact.

- *Use an adapter to bind Tasks to ListView rows.* A partial implementation of TaskAdapter is given to you in TaskList. You will need to complete the getView method, which returns the view to be displayed at a given position. View v will be an instance of the built-in layout `android.R.layout.simple_list_item_multiple_choice` - all you need to know about this layout is that it contains a CheckedTextView with ID `android.R.id.text1`. You should set the text of this CheckedTextView to the title of the task at array index `position`. Then, if the task is marked as completed, you should do the following:

  - Set the item as selected in the ListView, which will take care of marking the CheckedTextView as checked. You should use `ListView.setItemChecked` to do this.
  - If the item is checked, put a line through the task. This can be accomplished by using the `setPaintFlags` method of the CheckedTextView, and the `Paint.STRIKE_THRU_TEXT_FLAG` flag. The flags are just bits that can be turned

on and off; recall that you can turn on a bit by OR'ing it and turn off a bit by AND'ing the negation of the bit. You should make sure that when you change this bit, you preserve the existing paint flags; if you don't do this, the items in the list will not appear correctly. You can get the current paint flags for a CheckedTextView by calling the `getPaintFlags` method.

- You should then create an instance of this adapter and set `mTasks`' list adapter to this instance in `onCreate`. The arguments to the TaskAdapter constructor are a context, the resource ID of the view to display for each row (again, `android.R.layout.simple_list_item_multiple_choice`), and the array of Tasks which will be displayed in the ListView, which is defined as taskArray in TaskList.

- *Create the TaskEdit Activity.* Compared to TaskList, this Activity is fairly simple - all you need to do is initialize the UI to the layout that you wrote for the activity, and set the values of the fields to those that are passed in the input Intent. The `Intent.get*Extra` methods will allow you to recover the extras that were passed in.

- *Update AndroidManifest.xml.* Finally, you should update the manifest so that the TaskList activity will appear in the launcher, and so that the TaskEdit activity is available. You should set the label of this activity to the string `edit_task`.

Some Tips

The above list may seem daunting, but each of the items should take at most around 20 lines of code to complete. While you won't have to write too many lines of code, this assignment may still be challenging because of the difficulty of adjusting to a new platform. You should start as soon as possible to make sure that you understand all of the concepts needed to complete it, and come to office hours early if you get stuck.

You should also check the newsgroup frequently. Obviously, you may post questions if you need clarification; you may also find it helpful to keep an eye on the newsgroup even if you don't have any questions, as your classmates may point something out that you hadn't considered.

If you find any errors in the writeup or skeleton code, please post them to the newsgroup or email the instructor as soon as possible.

Karma Problems

See the course website for an explanation of what these problems are. If you complete any of them, include a readme.txt file in your submission explaining what you have completed. If there's something else you'd like to try to do for karma credit, you can e-mail the instructor for approval and guidance.

- Make the UI look nicer! The black screen and plain layout aren't the greatest looking of interfaces. Without modifying the basic layout too much, can you make Tasks a bit more pleasant to look at?

- Write some JUnit tests to test your UI. One possibility would be to try different arrays of Tasks and make sure that the UI is updated accordingly. You would need to add support for injecting a custom array into the Activity, and then use ActivityInstrumentationTestCase2 to set the array, launch the activity, and access the ListView to verify that the contents are correct.

- Add support for gestures. For example, if the user swipes horizontally over a task, the task could be deleted. Since deleting tasks isn't actually supported in the UI, you could write the code which gets the event without actually deleting it.