Project 2C – DreamCatcher Complete

30/30 Points

4/8/2022

Attempt 2

REVIEW FEEDBACK
4/5/2022

Attempt 2 Score:
30/30

View Feedback

Unlimited Attempts Allowed

Details

Due: Friday, April 8
Points: 30 points
Deliverable: ZIP file (30 points worth of tests)
Resources:

- Same build.gradle that was used in 2A and 2B
- BaseDreamCatcherCompleteTest (https://drive.google.com/file/d/1su_pAQp0KSeCmEc14_rkQe1u-nfIm_UX/view?usp=sharing)

Overview

In this final DreamCatcher project you will put finishing touches on your application:

1. Add a New Dream
2. Add a New Reflection (Dream Entry)
3. Implement Recycler-View for Dream Entries
4. Swipe to Delete Dream Entries
5. Share a Dream
6. Take a Photo

Preparation

Start a new Android project for this assignment and copy over any files from Project 2B that you feel are necessary. Alternatively, you can duplicate your Project 2B folder and use that to create you Android Project 2C.

1. Add a New Dream

In this part of the project you will add an App Bar to the DreamCatcher application. In the list view, the app bar will contain two menu items:

- An "Add Dream" menu item that uses a plus (+) icon to add a new dream. Choosing this menu item will add the dream to the database and take you to the detail screen, you can then edit the dream and press the back button to get back to the list view.
- A "Delete All Dreams" menu item that uses a trash can icon to delete all dreams from the database. Choosing this menu item will clear the database, remove both dreams and dream-entries.

This portion is very similar to how the app bar works for crimes in the CriminalIntent application, and is explained well in chapter 14 of BNR and the chapter 14 slides.

Naming requirements for this portion:

- The menu resource file should be called **fragment_dream_list**. Note that this will not clash with the layout named fragment_dream_list since it will be under the menu folder.
- The menu item that adds a dream should have ID **add_dream**.
- The menu item that deletes all dreams from the database should have ID **delete_all_dreams**.

Run you code and see if you can create a new dream now.

2. Add a New Reflection (Dream Entry)

In your detail view, you will add a floating action button (FAB) that allows you to create a new *reflection* (dream entry) for the current dream. A *reflection* is a kind of dream-entry that displays user-specified text and the date that the reflection was created. When the button is clicked, it creates a dialog box that allows you to enter your text. Creating the text-entry dialog is similar to creating a date dialog, which is described in chapter 13 of BNR. However, instead of putting a DatePicker in the layout, you will create a simple linear layout with an EditText view. For this portion of the assignment, use to the AddReflectionDialog class and the KeyboardUtil class. A good strategy for this section is:

1. Add the floating action button to your layout and make sure it appears

2. Create the alert dialog layout, and wire up the button so that it shows the alert dialog
3. Put the logic that controls the response to the dialog in the detail fragment
4. Once the entry has been added to the model, refresh the entries in the view

You do not need to update the database after you add the reflection. Continue to let the onPause callback update the database. Also, we recommend that you do **not** try to implement the recycler view for entries before you get this functionality working. Test the behavior with only the 5 entry buttons and make sure it works, then you can worry about the recycler view.

Naming requirements for this portion:

- Your floating action button should be in your fragment_dream_detail layout and it should have the ID **add_reflection_button**.
- The layout for your alert dialog should be named **dialog_add_reflection**.
- The EditText in your dialog should be named **reflection_text**. You may give this field a hint, but do not give it default text.

Use the following code for AddReflectionDialog. See the criminal-intent dialog slides (the last slide) for notes on the implementation.

```
class AddReflectionDialog : DialogFragment() {
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        val ui = DialogAddReflectionBinding.inflate(LayoutInflater.from(context))
        val okListener = DialogInterface.OnClickListener { _, _ ->
            parentFragmentManager.setFragmentResult(REQUEST_KEY_ADD_REFLECTION,
                bundleOf(Pair(BUNDLE_KEY_REFLECTION_TEXT,
                              ui.reflectionText.text.toString()))
            )
           }

        return AlertDialog.Builder(requireContext())
            .setView(ui.root)
            .setTitle("Add Reflection")
            .setPositiveButton(android.R.string.ok, okListener)
            .setNegativeButton(android.R.string.cancel, null)
            .create()
    }
}
```

In addition to the behavior described above, ensure that the FAB button is disabled when the dream is *fulfilled*. Therefore, once a dream is fulfilled, we cannot add a reflection after it. However, clicking the FAB when the dream is deferred *will* bring up the dialog box and allow you to add a reflection after it. The reflection will always go at the *end* of the dream entry list. Note that unchecking the dream deferred checkbox when there is a reflection after it, will make the deferred button disappear, but all reflections (even those that came after the deferred button) will remain.

Now that we have the ability to create new dreams and dream entries, remove the code from the DreamRepository that automatically generates 100 new dreams. Do this by removing the loop that creates and adds the dreams to the database: `for (i in 0 until 100) { ... }`. However, do *not* delete the code that deletes all dreams and dreamEntries from the database. This code is still needed so that we can reset the database for the testing suite.

Run your code and see if you can create a new reflection in the dream detail view. You should not be able to create a new reflection after the dream has been fulfilled, but you *should* be able to create one after the dream has been deferred.

[See Demo from the Q&A for how this should work at this point.]

3. Implement Recycler-View for Dream Entries

You will need to implement a recycler view for your dream entries. You must implement the Holder and Adapter as inner classes of DreamDetailFragment, similar to what we did with dreams in the list-view. You should rely heavily on chapter 9 of BNR, as well as the Adapter, Holder, and RecyclerView that you already implemented in the list-view portion of your application. If you want to refresh your memory on how we implemented the recycler-view for the criminal intent application, you can check out the Kaltura recording for Module 5's Q&A, and look at the list-detail slides (also in Module 5) starting around slide 35.

Follow these steps to help you implement your recycler-view for dream entries. Note that the names in bold are required.

- Create a layout for a dream entry item named **list_item_dream_entry**.
- Place a single button inside your layout with the ID **dream_entry_button**.
  - Caution: Ensure the button layout height is wrap_content, otherwise only one button may appear on your screen
- Replace the 5 buttons in your layout **fragment_dream_detail** with a recycler view with ID **dream_entry_recycler_view**.
  - Caution: Ensure the recycler-view height is 0 dp (match constraint), otherwise it will block other views
- Remove references to the buttons from DreamDetailFragment, but keep most code for reuse purposes
- **Run your app. You won't see buttons but dream entries should still be updated in memory and in the database**
- Create DreamEntryHolder in DreamDetailFragment. The bind function should call updateEntryButton or refreshEntryButton, assuming that you created such a function. If not, create it now – it should take two parameters, a button and a dream entry, and it should update the button based on the entry model.
- Create DreamEntryAdapter in DreamDetailFragment. It behaves similar to the adapter in DreamListFragment.
- Declare and initialize an adapter. Decide where to attach the adapter to the recycler view
- **Run your app. Everything should work**

The remainder of this project description is under construction. What follows should give you an idea how to proceed, but some of the details are out-of-date (like the method of taking a photo) and you will have to wait until next week for update information.

4. Implement Swipe to Delete for Dream Entries

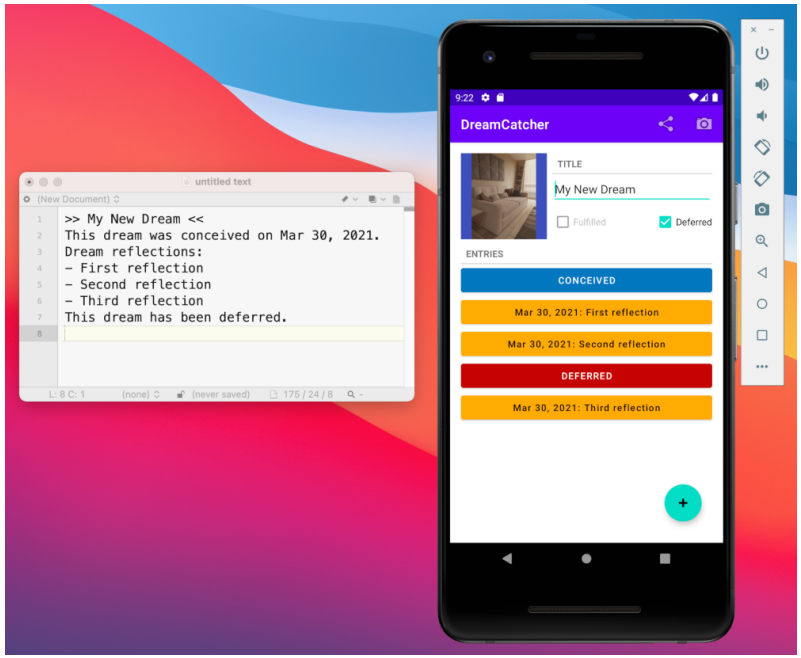
To implement the recycler view swipe to delete, **this article (https://medium.com/@zackcosborn/step-by-step-recyclerview-swipe-to-delete-and-undo-7bbae1fce27e)** may be helpful. Do not allow a swipe right to delete a dream entry, only a swipe left. You can limit the direction using programming logic, or use the getSwipeDirs function (not mentioned in the article). Also, you should only allow reflections to be deleted, do not allow conceived, fulfilled, or deferred buttons to be deleted. It's okay if the swipe action can be done on these buttons, but they should not disappear. Note that you do not need to add a background and icon, and you do not need to implement an undo snack-bar. You *may* do these things as challenges if you like, but it won't effect your grade. If you do intend to try these optional functionalities, make sure you get the swipe to delete working without them first. The article may do a few things differently than we do, so ask on Piazza for guidance.

5. Share a Dream & Taking a Photo

Sharing a dream and taking a photo will both be menu items in the detail view. Creating the menu items will be similar to creating the menu items that we did for the list view above. The functionality of sharing the dream is similar to "sending a crime report" in Chapter 15 of BNR.

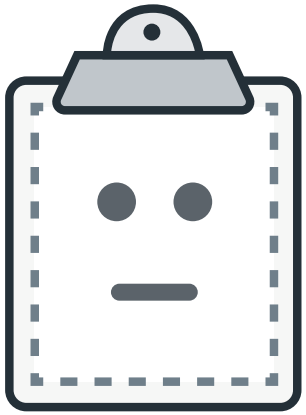
Instead of using startActivity with an Intent to take a photo (as mentioned in Chapter 16 of BNR) please use an activity-result-launcher, as shown in the slides. In both cases you will have to make adjustments due to the fact that your actions will be done using menu items rather than buttons.

Sharing a dream should look like the following. The text here obtained by clicking the share button in the menu bar and hitting "copy" when the application chooser came up.



Naming requirements for these portions are:

- The menu resource file should be called **fragment_dream_detail**. As above, this will not clash with the layout named fragment_dream_detail since it will be under the menu folder.
- The menu item that shares a dream should have ID **share_dream**.
- The menu item that takes a photo should have ID **take_dream_photo**.
- You must add an image-view to your fragment_dream_detail layout with ID **dream_photo**. The image view should be between 120dp square and 150dp square. Note that this bullet refers to the layout, not the menu. Also not that you will have to move elements around in your dream detail layout. Make sure you maintain good alignment when you do.



Preview Unavailable

DreamCatcher-3.zip

Download

(https://canvas.vt.edu/files/22713321/download?download_frd=1&verifier=Kan0e7ZImetu7IRB0F1NDvztHf7QhUOFFzcSNvlW)

You are unable to submit to this assignment as your enrollment in this course has been concluded.