CS 371M: RecyclerView

Recall the rules for flipped classrooms: do the work alone or with a partner that is unique for the semester. You can find the github link on Ed.

Only one student needs to submit this code, but both students may do so if they wish. This assignment should be submitted through github (sorry, partners can't share the repository). Include a README at the top directory level that contains both student's EIDs.

For this assignment, you will have two RecyclerView layouts, though only one of them is visible at a time. (This is not a good way to do things, but it will go away when fragments arrive next week). Both RecyclerViews hold a list of color names, with the background color matching the text. So the background of the entry for "blue" is blue.

The RecyclerView called `recyclerViewLinear` will display a linear list of colors, each of which spans the entire width of the screen. The Recyclerview called `recyclerViewGrid` will display a list of colors, three across, using bubbles (or "cards") that are about $\frac{1}{3}$ the width of the screen.

In the AppBar, there is a swap icon that you can click to move back and forth from the linear to the grid recyclerviews (once you provide needed code).

As far as dynamic behavior, when you click a color in the linear view, it moves to the top of the list (which might be off screen). When you click on the color in the grid view, it swaps with a randomly chosen **other** color (don't swap with the same color). You can see this in action in the demo video (linked off the course schedule web page).

They layout for a color is in `color_card.xml`. You don't need to change it, but you can take a look at what it does and judge for yourself if you find the result pleasing on the eye. As far as the text content, your code should print the luminance of the color in addition to its name. And if the luminance is less than 0.3, print the color name in white, otherwise print it in black. There is a comment in the code showing you how to format and print the text.

There is code in the project to compute luminance. You can read about luminance and the HSL color space here.
`https://en.wikipedia.org/wiki/Luminance`
`https://en.wikipedia.org/wiki/HSL_and_HSV`

Everything that you need to do is marked in the code this way:
`// XXX Write me`
There might be hint code in a comment after the "Write me" comment. If you do not see a XXX, you do not need to change the code and you probably should not change the code.

Once you write code for MainActivity and ColorAdapter, your RecyclerView should work.

1. **Files of interest**

   (a) **AndroidManifest.xml** All good.

(b) **MainActivity.kt** Write `initRecyclerViewLinear`. You can pattern match from `initRecyclerViewGrid` (which is a function that is provided) and from the Demo code I gave you. Use the ColorAdapter.

(c) **ColorAdapter.kt** There is a lot going on in this file, so take some time and let's look at it.

First off, the class declaration looks like this.

```
class ColorAdapter(private val colorList: List<ColorList.ColorName>,
  private val onClickListener: ColorAdapter.(Int) -> Unit)
  : RecyclerView.Adapter<ColorAdapter.VH>() {
```

The use of "private val" means that in addition to being a parameter, we are declaring `colorList` to be an instance variable. Same with `onClickListener`. Both of these are member variables, initialized by the parameters (but without having to write the initialization explicitly). Very cool, Mr. Kotlin!

This `onClickListener` thing is a strange beast. It is a function, which explains the `(Int) -> Unit` part of the type signature. This signature says it is a function that takes an integer and returns nothing. The `ColorAdapter.` prefix says that this is a (pointer to a) member function. A member function has access to instance variables.

Look in `initRecyclerViewGrid` for an example of how we pass a pointer to a member function (hint, Class::functionName).

Check out how we initialize the member variable `list`. Kotlin provides this neat `apply` function that we use. Then in the body of the lambda, we are calling the `addAll` member function of `MutableList`.

See also the inner class VH, which stands for view holder. An inner class is a cool idea, not every language supports nested classes. Also note that ColorCardBinding is just a view binding object created from `color_card.xml`.

In `onCreateViewHolder` notice that we can create a view holder (VH) object using a binding object. You should set an on click listener for the root of this binding. The root view of a binding object contains all other views nested within it. The onclicklistener will call your `onClickListener` member function, but that takes an integer parameter, and you don't have one yet. How will you get one?

`onBindViewHolder`. Please write the guts of this function. It gets the object at the given position, computes the luminance, writes the color name and luminance into the TextView using black or white according to the rule above. Also, please set the background color for the TextView to the color being named.

`moveToTop`. Remove the list member at index `position` and insert it at the start of the list. You then have to notify the system to redraw the display. I thought I could use `notifyItemRemoved` and `notifyItemInserted`. But I could not get that to work. So start with `notifyDataSetChanged` and once that works you can see if you can find a more efficient alternative (because `notifyDataSetChanged` just blindly redraws everything instead of redrawing the minimal amount).

`swapItem`. There is some basic logic missing here. Recall that the logic manipulates the state of your list. If you expect to see the result of this manipulation, you need to tell the adapter that something has changed. We talked about `notifyDataSetChanged`. That is one option and maybe a good one for debugging. I was able to get multiple calls to `notifyItemChanged` to work just like `notifyDataSetChanged`, but it is more efficient.

One hint here. Instead of starting with choosing a random other color, start by choosing the color at index 0. That will allow you to see both colors get swapped. But don't stop here. The specification is to swap with a randomly chosen other color. I just suggest 0 as a way to gain confidence that your implementation is correct.

(d) **color_card.xml** This file contains the layout of a row. I tried to fancy it up a bit. No changes needed.

(e) **ColorList.kt** No changes are needed, but you might want to look at it. Thanks to the nut jobs on the net who have cultivated this list over the years.