Stanford University, CS 193A
Homework 4: Friendr
(Pairs allowed)

Assignment idea and spec by Marty Stepp.

This assignment practices topics from the past few weeks such as fragments, styling, and Android libraries.

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.

Starter Files:

- friendr-starter-files.zip (ZIP of all starter files)
- friendr_logo.png (logo image)
- friends_theme.mp3 (background music)
- FriendrMainActivity_partial.kt (partial solution to `FriendrMainActivity`)
- activity_friendr_main_partial.xml (partial solution to `activity_friendr_main` layout)
- Other images and resources are found in the ZIP file above. Download it!

File and Project Names:

- Your **Android Studio project**'s name must be EXACTLY the following, case-sensitive, including the underscores:
  - `cs193a_hw4_`***`SUNETID`*** (example: `cs193a_hw4_ksmith12`)

  Or if you are working in a pair, please list both partners' SUNetIDs separated by an underscore:
  - `cs193a_hw4_`***`SUNETID1_SUNETID2`*** (example: `cs193a_hw4_ksmith12_tjones5`)
- You must have activity classes with the exact names given below under "Program Description," case sensitive, in corresponding file names (e.g. class `FooActivity` in file **FooActivity.kt**. You may add other activities, classes, and files to your app if you like, but

you must have the exact activity file names below at a minimum, spelled and capitalized exactly as written here.

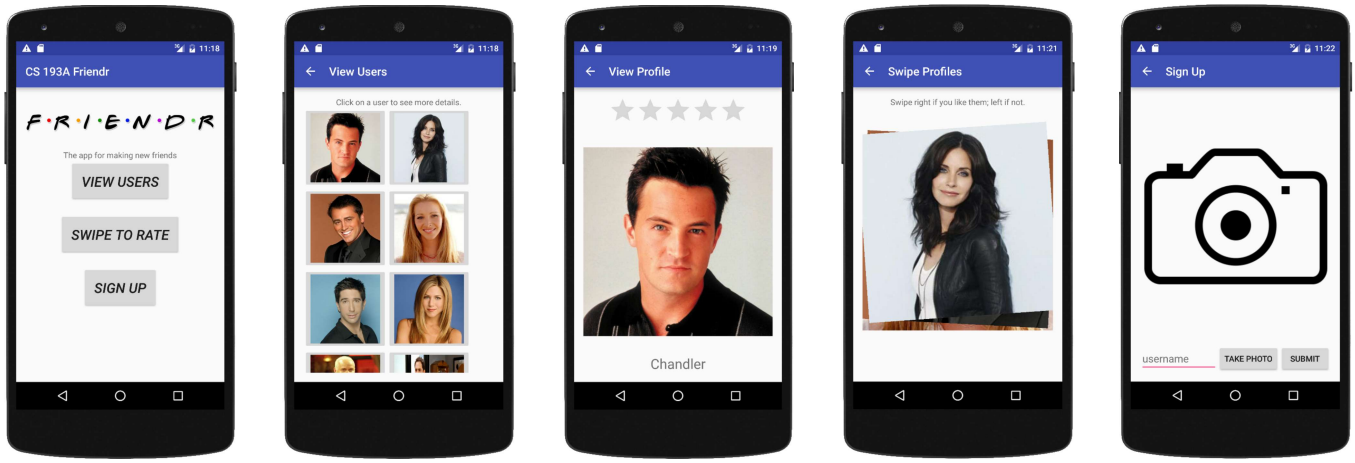
- When you are finished, ZIP your project (instructions) into a file with the following exact file name, including spelling and capitalization:
  - `cs193a_hw4_SUNETID.zip` (example: `cs193a_hw4_ksmith12.zip`)
  - or `cs193a_hw4_SUNETID1_SUNETID2.zip` if you work in a pair (example: `cs193a_hw4_ksmith12_tjones5.zip`)
- If you use any **libraries** in your app, make sure they are located in **app/libs/** and are included in your ZIP that you turn in. If you use libraries from the Maven repository (ones that are added to a project by declaring them in your **build.gradle** file), these will be auto-downloaded when we compile your project to grade it.
- Turn in link: Turn in HW4 here.

Program Description and Functionality Requirements:

For this assignment you will write an app named **Friendr** for finding and rating friends, where the interface is a lot like an "online dating" app such as Tinder, OKCupid, or Match. The app's name is a play on the idea of it being for "friends" + "Tinder", along with the fact that one of our data sets comes from the 1990s TV show, *Friends*. (You do not need to use the *Friends* data set nor name your app "Friendr" if you don't want to; we have some other data sets you can use instead, or you can make your own, as described later in this document.)

We would like you to write the following activity classes:

- `FriendrMainActivity`: An initial "welcome" screen with links to other activities.
- `ViewUsersActivity`: Displays all of the people available in the app.
- `ProfileActivity`: Displays details about an individual person and allows the user to rate them.
- `SwipeActivity` *(optional)*: Allows the user to rate people by swiping left and right.
- `SignUpActivity` *(optional)*: Allows the user to sign into the app and/or sign up for an account.



`FriendrMainActivity` `ViewUsersActivity` `ProfileActivity` `SwipeActivity` `SignInActivity`

Screenshots of app's various activities

You do not need to exactly match the appearance of the screenshots in this document. Please feel free to be creative and customize your app as you like, so long as your app contains the functionality described below.

FriendrMainActivity:

Your `FriendrMainActivity` acts as the main hub of the app; when the user presses the Back button, they return to this activity. It must have the following features:

- **Logo:** Shows a logo for the app, such as the one provided in our starter files as **friendr_logo.png**.
- **Description Text:** Shows some text explaining the program to the user.
- **Buttons:** A set of buttons (or some reasonable widget) that provides a way to go to the various other activities. At a minimum, the user should be able to go to the `ViewUsersActivity`, but you should probably also include a way to get to other activities like `SwipeActivity` or `SignInActivity` if you are writing those.
- **Audio:** We want your app to play some background music or other audio using Android's `MediaPlayer`. The music should play when the activity loads up, and it should pause when you exit this activity to go to another activity. If you return to the `FriendrMainActivity` later, the music should resume from where it left off. In our starter files we provide an MP3 version of the *Friends* theme song called **friends_theme.mp3**. You can put it in the **res/raw** folder of your project.

Since the `FriendrMainActivity` mostly consists of old material that is not core to HW4, we have provided a **partial solution** to it that you can use as starter code. We have posted links above to FriendrMainActivity_partial.kt and activity_friendr_main_partial.xml that contain partial solutions for the Kotlin code and XML layout for this activity, respectively. If you want to use this partial code, click these links to see the code, and then copy/paste portions of it into your own project's files. The files provided here are partial and don't work on their own without further modifications.

ViewUsersActivity:

Your `ViewUsersActivity` will show a list of all people in the system. It must have the following features:

- **Photos:** A photo of each user in the app's online data set.
- **Click to View Profiles:** A way to click on that user to view that user's profile, to go to the `ProfileActivity` about that user. *(See section later about landscape mode.)*
- **Layout:** You should use a reasonable layout to display the list of people. At a minimum, the list of users should be scrollable so that the user can see all the people, no matter how many people are present in the data set. The images should show up at a reasonable size; for example, the images should not be too tiny to see each person's face, nor should they be so large that you cannot see multiple people on the screen at once. Our screenshot uses a two-column `GridLayout`, but you can use whatever reasonable layout you feel is best.
- **Rotation:** Your `ViewUsersActivity` must use a different layout when in **portrait** vs. **landscape** mode, as described later in the `ProfileActivity` section.

Unlike in some past assignments where the images were located as resources inside the app, in this assignment the resources are located on the web. Part of the point of this assignment is to practice writing an app that dynamically **downloads** image resources from the web using a library such as Picasso. You will not receive full credit on the assignment if you download the user profile images ahead of time and put them in your project's **res/drawable** folder; do not do this.

The following URL contains a list of people names:

- http://www.martystepp.com/friendr/friends/list

The text data at that URL contains the people's names, 1 per line, in the following format:

```
Chandler
Monica
Joey
Phoebe
Ross
Rachel
```

example contents of friends list URL (abridged)

Each line corresponds to an image file that your app can fetch; for example, the line "Chandler" corresponds to the following image file, **chandler.jpg** (notice that the "C" in the file name is in lowercase). For some data sets, there may also be a .txt file with the same name that contains additional information text about that person, which you may want to fetch using Ion and display on the screen.

- http://www.martystepp.com/friendr/friends/chandler.jpg (picture of Chandler)

- http://www.martystepp.com/friendr/friends/chandler.txt (text information about Chandler)

If you're using Picasso, that library lets you set a placeholder image for while an image is downloading, and an error image in case the image is unable to be loaded. We provide some images that you might want to use for that in our starter ZIP, named **loading.gif** and **sadface.png** respectively.

In your `ViewUsersActivity`, you must fetch the list of users (using the Ion HTTP library shown in class, or a similar library) and use this to help you fetch the images for all users (using the Picasso library shown in class, or a similar library) and display them on the screen. Since you don't know ahead of time how many users there will be, you will have to dynamically create and add the images (in our screenshot, we display them as `ImageButton` objects) to the screen in your Kotlin code.

If you want another data set for testing purposes, try changing the **/friends/** in the list URL to **/pokemon/** or **/bachelor/** .

Tip: To make the user images look good on the screen, you may want to resize them to some fraction of the overall screen size. It may help you to get the device's screen size. In standard Android, you can write:

```
1 // get the screen width and height in pixels
2 val display = windowManager.defaultDisplay
3 val size = Point()
4 display.getSize(size)
5 val width = size.x
6 val height = size.y
7 ...
```

ProfileActivity:

Your `ProfileActivity` must have the following features:

- **Photo:** A large photo of that person.
- **Name:** The person's name.
- **Information** (optional): Our data set contains text descriptions/profiles of each person. If you want to display that text here, you may, though it is not required. See the section below for details about our data set.
- **Rating:** Provide a way for the current user to "rate" this person. Exactly how you do this is somewhat up to you. In our screenshot, we use a `RatingBar`, a widget we haven't discussed much in class, but you can read documentation for if you are interested. You can place a `RatingBar` into a layout and interact with it in Kotlin code by calling its `getRating` and `setRating` methods. If you want to trigger an event when the user clicks the stars in the rating bar, use the rating bar's method `setOnRatingBarChangeListener` method; or, for simpler coding, you can have a separate "Submit" button that sends the user back to the `ViewUsersActivity`.

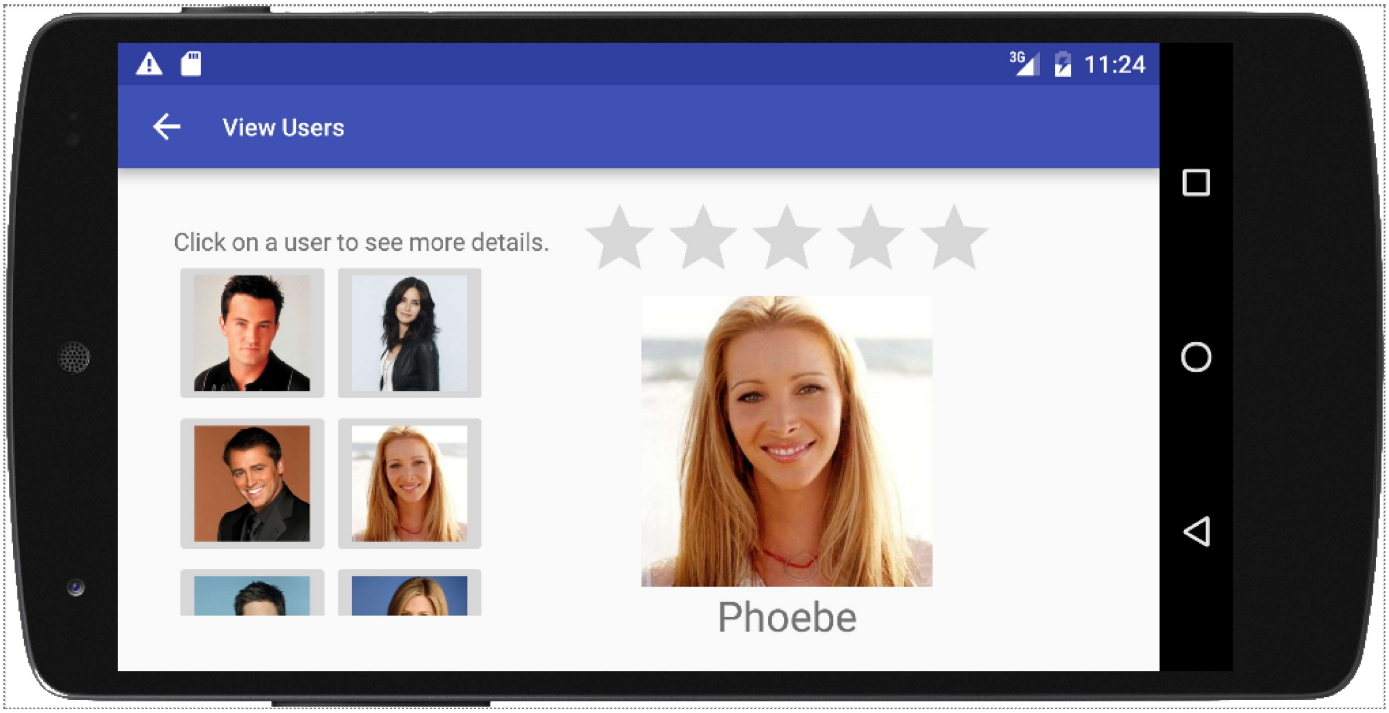
If you don't want to use a `RatingBar`, you can just provide a simpler more familiar UI, such as two buttons, one saying "Like" and one saying "Dislike", or whatever system you prefer, so long as it lets the user indicate some kind of rating of the user. Another option is to make the profile "swipe-able" using a library of your own choosing.

- **Persistent (saved) Ratings:** Whatever UI you use to allow the user to rate the person, you must save the rating that the user gives each person so that if they return to this profile later, the rating will still be shown. (You could use a file with a `PrintStream` as in HW2, or you could use `Preferences` as shown in lecture.)

Before you start to implement `ProfileActivity`, read the next section about **landscape mode** because it contains important information about how fragments should be used in your `ProfileActivity`.

Landscape Mode:

Most of your app can use the same behavior and layout in both portrait and landscape orientations. But you should modify your `ViewUsersActivity` so that in landscape mode, when the user clicks on any person's image, the person's profile is shown inside the same activity. It should look something like the screenshot below:



To accomplish this, make another layout for the `ViewUsersActivity` that goes in the **layout-land** folder, as shown in lecture.

You should work to avoid redundancy in your code for viewing users and profiles. The recommended way to do this is to put the actual bulk of the layout, code, and logic from this activity into a `ProfileFragment`. (This is what we do in our own instructor solution.) The

`ProfileActivity` would then just be a thin wrapping around this fragment that initializes its state by telling it which user to display. Then in your `ViewUsersActivity`'s landscape mode layout, you can place a `ProfileFragment` in the right-side area. When the user clicks a user icon at left, the activity should talk to its fragment and tell it what user to display without changing activities.

If you don't want to use a fragment, another option for avoiding the redundancy is to use a **custom layout XML file** and the **layout inflater** as taught in class. You must choose one of these two options (fragment or XML/inflater) and incorporate it into your code. Our general recommendation is to use the fragment, but it is up to you.

SwipeActivity: *(optional)*

Our app lets the user give people 5-star ratings, but more modern apps like Tinder prefer "swiping" left or right to indicate liking or disliking them. If you want to try an optional activity with this kind of behavior, make a `SwipeUsersActivity` that utilizes a library to "swipe" on the users. If you want to make the swiping compatible with the normal `ProfileActivity`/`Fragment`, perhaps you should consider a swipe-left to be a 1-star rating and a swipe-right to be a 5-star rating. This is optional and not required.

SignInActivity: *(optional)*

You are not required to store any personal information about the current user using the app, such as their name or photo. But if you want to make an optional activity that lets the user sign in to the app, and/or create a user profile in the app, you can create a `SignInActivity` to do so. Consider using the Google Sign-In Library shown in class to do this, or you can write your own sign-in feature if you prefer. This is optional and not required.

Extra (Optional) Features:

If you are interested in adding more functionality to your app beyond what is required, here are some options

- **Use Animations library::** Try using the Android Animations library to cause your app to animate when the user picks a profile, supplies a rating, etc.
- **Use other libraries:** Go find a library that we have not seen in class, and try incorporating it into your app.
- **Create Profile with photo:** If you want to practice using the Android camera, you could make an option for the user to create a profile for themselves and let them take a photo of

themselves as part of the profile creation. You would need to save this photo to a file so that it won't go away when the app restarts.

- **Gender preferences in matches:** Some dating sites ask the user what gender(s) they are seeking and then show only profiles that match those gender(s). For example, if the user is a female seeking males, only male profiles are shown in the list. This is a sensitive feature, because we don't want to force everyone into a gender binary nor exclude anyone's particular preferences. It is just listed here as an idea for additional functionality, filtering down the profiles based on some criteria given by the user. You are of course welcome to implement any variant of this idea, such as an LGBT version of the app, etc.

Style Requirements:

(These are the same as in Homeworks 1-3.)

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 specific 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 4
 * 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.