Toolbar and Soo Greyhounds Basic UI-13

In this assignment, you will start the Soo Greyhounds app by creating the skeleton/template of the app's layout, as well as a menu that will be displayed in the toolbar. This menu will have actions for users to accomplish tasks within the app. You will also enable the Up button in the toolbar.

1. Open Android Studio and create a new app using an empty activity. Name the app "Soo Greyhounds Mobile" and also update the package name to be "com.soogreyhounds.soogreyhoundsmobile". Set the API level to 19.

2. Create a new repository on GitHub, add your instructor as a collaborator. Then go over to Android Studio, attach your remote repository, and do an initial commit/push.

3. The first thing we will do is setup our screens for the entire app. We already have an activity created by Android Studio for the main screen (MainActivity.java). Create a new empty Activity for each of the following:

   a) PhotoDetailActivity
   b) PhotoViewerActivity
   c) LoginActivity
   d) EditProfileActivity
   e) CreateAccountActivity

4. We are now going to setup a toolbar to be used within our app. Your API level is 19 and newer, which means that you cannot use the native toolbar on all supported versions of Android. Luckily, the toolbar has been back-ported to the AppCompat library. The AppCompat library allows you to provide a Lollipop'd toolbar on any version of Android back to API 9 (Android 2.3). Open AndroidManifest.xml and look at the application tag. Notice the android:theme attribute.

   ```
   <application
     android:allowBackup="true"
     android:icon="@mipmap/ic_launcher"
     android:label="@string/app_name"
     android:supportsRtl="true"
     android:theme="@style/AppTheme" >
   ```

5. The AppTheme is defined in res/values/styles.xml. Open this file and ensure that the parent theme of your AppTheme matches the shaded portion shown below. For now, do not worry about the attributes inside the theme. You will update those soon.

   ```
   <resources>
     <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
       <!-- Customize your theme here. -->
       <item name="colorPrimary">@color/colorPrimary</item>
       …
     </style>
   </resources>
   ```

6. Let's work on our menu. Your menu will require a few string resources. Add them to strings.xml as shown below. These strings may seem mysterious at this point, but it is good to get them taken care of. When you need them later, they will already be in place, and you will not have to stop what you are doing to add them.

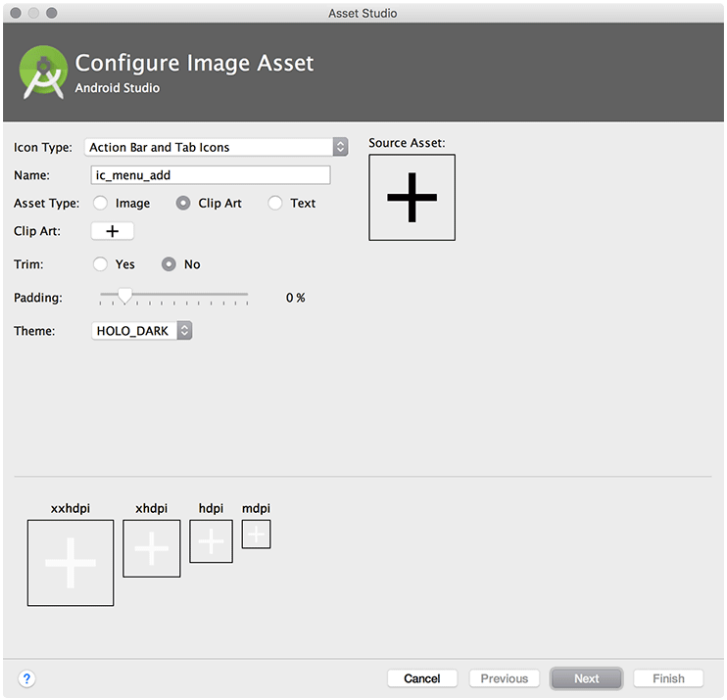
```
<resources>
   <string name="app_name">Soo Greyhounds Mobile</string>
   <string name="login">Login</string>
   <string name="create_account">Create Account</string>
   <string name="edit_profile">Edit Profile</string>
   <string name="photo_viewer">Photo Viewer</string>
   <string name="photo_detail">Photo Detail</string>
</resources>
```

7. In the project tool window, right-click on the res directory and select New → Android resource file. Change the Resource type to Menu, name the menu resource activity_main, and click OK.

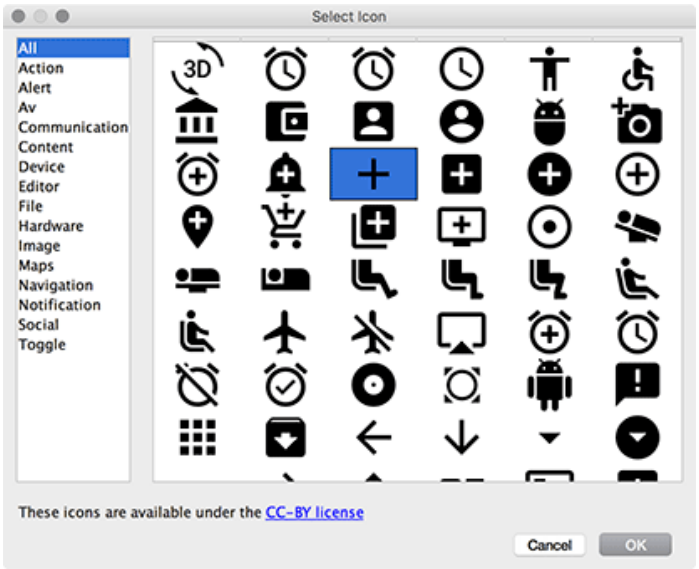
8. Here, you use the same naming convention for menu files as you do for layout files. Android Studio will generate res/menu/activity_main.xml, which has the same name as your MainActivity's layout file but lives in the menu folder. In the new file, switch to the Text view and add an item element as shown below. The showAsAction attribute refers to whether the item will appear in the toolbar itself or in the overflow menu. We have chosen never here to ensure the items only show in the menu drop down.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
   <item
      android:id="@+id/login"
      android:title="@string/login"
      app:showAsAction="never"/>
   <item
      android:id="@+id/create_account"
      android:title="@string/create_account"
      android:icon="@android:drawable/ic_menu_add"
      app:showAsAction="ifRoom|withText"/>
   <item
      android:id="@+id/edit_profile"
      android:title="@string/edit_profile"
      app:showAsAction="never"/>
</menu>
```

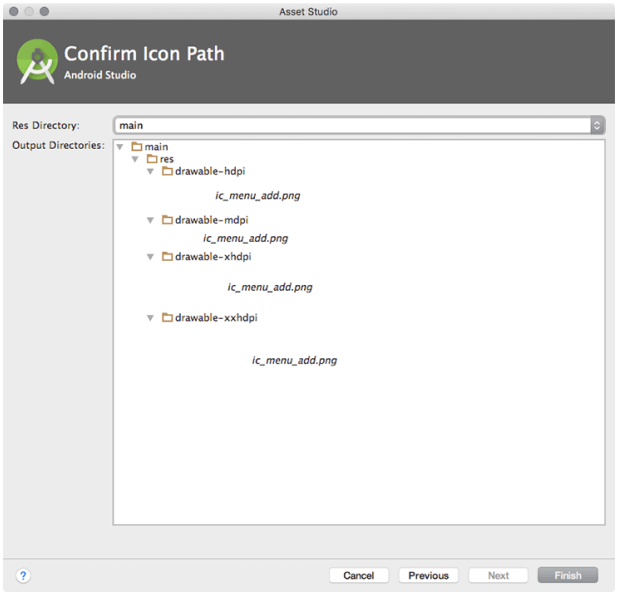
9. Right-click on your drawable directory in the project tool window and select New → Image Asset to bring up the Asset Studio. Here, you can generate a few types of icons. In the Icon Type field, choose Action Bar and Tab Icons. Next, name your asset ic_menu_add and set the Asset Type option to Clip Art. Update the Theme to use HOLO_DARK. Since your toolbar uses a dark theme, your image should appear as a light color. These changes are shown below; note that while we are also showing the clip art you are about to select, your screen will feature the adorable Android logo.



10. Select the Clip Art button to pick your clip art. In the clip art window, choose the image that looks like a plus sign.



11. Back on the main screen, click Next to move to the last step of the wizard. The Asset Studio will show you a preview of the work that it will do. Notice that an hdpi, mdpi, xhdpi, and xxhdpi icon will be created for you. Jim-dandy. Select Finish to generate the images.



12. In your layout file, modify your icon attribute to reference the new resource in your project.

```
<item
   android:id="@+id/create_account "
   ~~android:icon="@android:drawable/ic_menu_add"~~
   android:icon="@drawable/ic_menu_add"
   android:title="@string/create_account"
   app:showAsAction="never"/>
```

13. In MainActivity.java, override onCreateOptionsMenu(Menu, MenuInflater) to inflate the menu defined in activity_main.xml.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.activity_main, menu);
    return true;
}
```

14. Run the app to see your menu. Click the three dots to see your menu items.

15. Let's make our menu work! In MainActivity.java, implement onOptionsItemSelected(MenuItem) to respond to selection of MenuItems.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
      case R.id.login:
        Intent loginIntent = new Intent(this, LoginActivity.class);
        startActivity(loginIntent);
        return true;
      case R.id.create_account:
        Intent createAccountIntent = new Intent(this, CreateAccountActivity.class);
        startActivity(createAccountIntent);
        return true;
      case R.id.edit_profile:
        Intent editProfileIntent = new Intent(this, EditProfileActivity.class);
        startActivity(editProfileIntent);
        return true;
      default:
        return super.onOptionsItemSelected(item);
    }
}
```

16. We are going to see how hierarchical navigation works. First let's add a button to our MainActivity and a button to our PhotoDetailActivity so we can test the navigation (in a future lab we will expand this as we build more of this app). Start by doing the following:
    a) Add a button in your activity_main.xml layout file and set it's text to "Open Photo Detail"
    b) Give it an id of "photoDetailButton"
    c) Add a button in your activity_photo_detail.xml layout file and set it's text to "Open Viewer"
    d) Give it an id of "photoViewerButton"

17. Next, we will hook up our button so they open the relating screens. Open MainActivity.java and add the following code.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    …
    Button photoDetailButton = findViewById(R.id.photoDetailButton);
    photoDetailButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent intent = new Intent(getBaseContext(), PhotoDetailActivity.class);
        startActivity(intent);
      }
    });
}
```

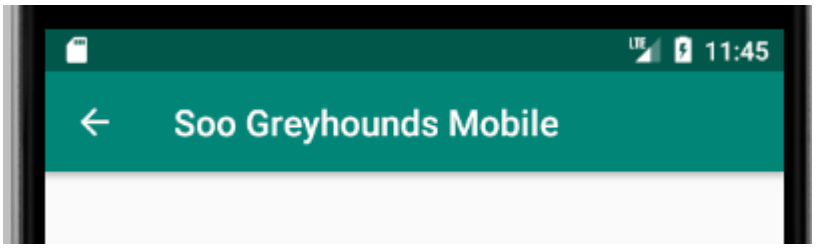
18. Now open PhotoDetailActivity.java and add the following code.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   …
   Button photoViewerButton = findViewById(R.id.photoViewerButton);
   photoViewerButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
         Intent intent = new Intent(getBaseContext(), PhotoViewerActivity.class);
         startActivity(intent);
      }
   });
}
```

19. Now, let's enable hierarchical navigation by adding a parentActivityName attribute in the AndroidManifest.xml file.

```
<activity
   android:name=".PhotoViewerActivity"
   android:parentActivityName=".MainActivity">
</activity>
```

20. Run the app, click the button you created on the main screen, then click the button on your photo detail screen. Notice the Up button on the photo viewer screen. Pressing the Up button will take you up one level in the app's hierarchy (not back one screen) to MainActivity.



21. Let's give our screens better titles. For now, we will reuse our menu strings, however as we build this app we will change this (some will have their own strings, some will be dynamically set). Open AndroidManifest.xml and updated the titles of each activity.

```
<activity android:name=".CreateAccountActivity"
  android:label="@string/create_account" />

<activity android:name=".EditProfileActivity"
  android:label="@string/edit_profile" />

<activity android:name=".LoginActivity"
  android:label="@string/login"/>

<activity android:name=".PhotoViewerActivity"
  android:parentActivityName=".MainActivity"
  android:label="@string/photo_viewer"/>

<activity android:name=".PhotoDetailActivity"
  android:label="@string/photo_detail"/>
```

22. Run the app again, click the button you created on the main screen, then click the button on your photo detail screen. Notice the different tiles in the toolbar.