CS370 Spring 2017 Homework 2

Homework 2.1 Obtaining Code

1. Ensure that the lab workstation is booted into OS X

2. Create a new directory on your desktop called Repositories.

3. Open a terminal session (Applications->Utilities->Terminal).

4. If you do not have a GitHub account, create one here: **https://github.com**

5. Using the command line, change directory to the Repositories directory that you've just created.

6. Ensure that there are no other default accounts entered into the OS X keychaing (keychain management instructions here: https://kb.wisc.edu/helpdesk/page.php?id=2197 . Search for the github.com entries and remove them).

7. Clone the repository located at https://github.com/SSU-CS370/Homework2-370.git

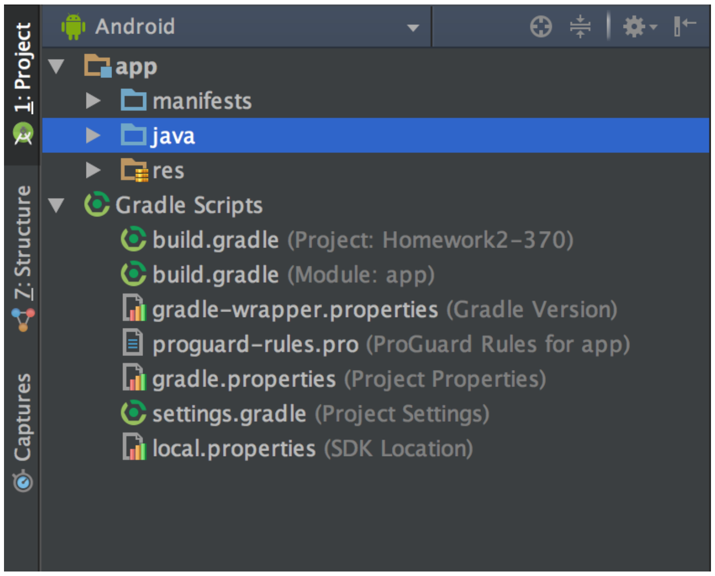
8. Branch the repository using a branch name of '**last-name + first-name + 370H2'**

9. Open Android Studio.

10. Open the Android project that you just cloned.

Honework 2.2: Android Deserialization

1. On the left hand side of the Android Studio Instance, click the 'Project' tab and ensure that the 'Android' view is selected. This will present you with the proper file navigation layout for the lab.


1: Project
Z: Structure
Captures
Android
app
manifests
java
res
Gradle Scripts
build.gradle (Project: Homework2-370)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

2. Expand app->java->com.example.jeff.homework2_370->activities. You should see a file called **SearchActivity**. Recall that Activities represent the code for the presentation layer in Android.

3. Expand app->res->layout. You should see a file called **activity_search.xml**. Recall that Layout.xml files represent the markup mechanism for Android apps, and that they are typically backed by an Activity.

4. Open the activity_search.xml file. Ensure that you are in Text view mode and not Design view mode. There are tabs at the bottom of the file window that allow you to switch between these views.

5. Note that there are several elements in this layout file:
    a. **LinearLayout** (the root layout for the view that determines how child elements are arranged)
    b. **TextView** (an element for displaying text)
    c. **Button** (an element allowing the user to invoke a functionality)
    d. **EditText** (an element allowing users to provide text input)

   e. **ImageView (**an element used to display an image)

6. Open the **SearchActivity** file. Note that objects corresponding to the layout elements in the activity_search.xml file have been declared as private variables.

7. Expand app->java->com.example.jeff.homework2_370->model. Note that there is a **RecipeModel** object defined. Add two new private String variables called **recipeImageUrl** and **recipeDescription**. Don't forget to provide a getter/setter for each (right-click on the class name, select 'Generate' and then 'Getter and Setter', select the two new variables and click 'OK') .

8. Expand app->java-> com.example.jeff.homework2_370->network. Note that there are three separate classes defined in this directory:

   RecipeCallbackListener (an interface that we will employ using the Listener pattern to handle responses from background threaded processes)

   RecipeSearchAsyncTask (a mechanism for assigning long running tasks to a background thread)

9. Expand app->java-> com.example.jeff.homework2_370->utility. Note that there is a single class defined in this directory:

   RecipeParser (a Singleton used to deserialize recipe JSON into a RecipeModel)

10. Return to the **SearchActivity** file. Starting on line 23, add new lines to the **onCreate** function that assign the instance of the layout element in the activity_search file to the corresponding private variables using the *findViewById* function (this should be straightforward as we have covered it in Lab1).

11. Next add a click handler for the Button object as we have in other labs (type the name of the button followed by .setOnClickListener and provide a new **View.OnClickListener** as an argument). If you have done this correctly, a new function called **onClick(View v)** should auto-generate. Leave it's internal implementation blank for now.

12. Add a **RecipeCallbackListener** private member variable called **recipeCallbackListener** to the **SearchActivity** class.

13. Returning to the **onClick** method in your **Button's** click handler, add this code (TYPING IT IN):

```
recipeCallbackListener = new RecipeCallbackListener() {
    @Override
    public void onRecipeCallback(RecipeModel recipeModel) {
        recipeName.setText(recipeModel.getRecipeName());
    }
};
```

This represents a concrete implementation of the **RecipeCallbackHandler** interface. If you open the **RecipeCallbackHandler** file, you will see that the interface only contains one method needing to be overriden. The inline instantiation that you should have just provided has an @**Override** directive (an annotation) for that method.

The body of the overridden method contains code that sets the text attribute of the **recipeName TextView** to the value of the **RecipeModel** passed in.

14. Beneath the block instantiating the concrete implementation of the **recipeCallbackListener**, add the following lines of code:

```
RecipeSearchAsyncTask task = new RecipeSearchAsyncTask();
task.setRecipeCallbackListener(recipeCallbackListener);
task.execute(searchEditText.getText().toString());
```

This code creates a new **RecipeSearchAsyncTask** for your background threaded operations, sets an instance of **RecipeCallbackListener** on the AsyncTask, and executes the doInBackground function.

15. Open the **RecipeSearchAsyncTask** file and add the following code to the doInBackground function above the 'return null' line:

```
String searchParams = params[0];

OkHttpClient client = new OkHttpClient();

HttpUrl.Builder urlBuilder =
HttpUrl.parse(HomeworkApplication.getInstance().getBaseApiUrl()).newBuilder();

urlBuilder.addQueryParameter("_app_key",
HomeworkApplication.getInstance().getApiKey());

urlBuilder.addQueryParameter("_app_id",
HomeworkApplication.getInstance().getAppId());

urlBuilder.addQueryParameter("your_search_parameters", searchParams);

String url = urlBuilder.build().toString();

Request request = new Request.Builder().url(url).build();

Response response = null;

try {
    response = client.newCall(request).execute();
```

```
        if (response != null) {
            return RecipeParser.recipeFromJson(response.body().string());
        }
    } catch (IOException e) {
        // do something with exception
    }
```

This code creates a new instance of an **OkHttp** library object called **OkHttpClient**, which is used to make http requests.

The **HttpUrlBuilder** is used to build an URL request string for the recipe API.

A **Request** object (also from the **OkHttp** library) is created to execute the request.

A **Response** object (also from the **OkHttp** library) is created to encapsulate the response form the remote API. The response is converted streamed into a string and assigned to local variable. We then call the *recipeFromJson* function on the **RecipeParser** Singleton and return a new **RecipeModel** if parsing is successful. If the operation is successful, the **RecipeModel** instance is returned to the main thread.

16. Open the **RecipeParser** class. Note that, per the comments, the response String that we received from the **RecipeAsyncTask** has been passed in and converted to a **JSONObject**. This object contains the entire JSON response. Farther down, the 'matches array is pulled from the JSON and assigned to a **JSONArray**. Lastly the code pulls another **JSONObject** form the **JSONArray** and interrogates the object to retrieve the appropriate properties.

17. Please enhance the parseRecipeFromJson method by also assigning properties to the **recipeImageUrl** property that you added to the **RecipeModel** object.
    a. You will first need to create another **JSONArray** and assign the "smallImageUrls" value from the recipeJson object.
    b. Next, interrogate the first index location of the array and assign to the **recipeImageUrl** of the **RecipeModel**.

18. Return to the **RecipeSearchAsyncTask**. In the onPostExecute method, invoke the local instance of **RecipeCallbackListener**'s *onRecipeCalback* method and pass it your response. This will send the RecipeModel instance back to the SearchActivity, where the value will be displayed on the app screen.

19. If your app is building correctly, open your terminal instance. From the Homework2-370 directory, execute the following commands sequentially:

    git add .
    git commit –m "working code complete"
    git push origin <your branch name>

Your code branch has now been saved and committed to the git repository.

This will complete the lab.