CS 370 Lab 4: JSON Deserialization

This lab is about manually deserializing JSON models.

Obtain Code

1. Ensure that the lab workstation is booted into OS X
2. Create a new folder on your desktop called Repositories
3. Open a terminal session (Applications -> Utilities -> Terminal)
    - This is just a terminal window on your local machine! ***Do not log in to blue!***
4. Ensure there are no other default accounts in the OSX keychain (if on public/lab computer):
    a. Keychain management instructions: https://kb.wisc.edu/helpdesk/page.php?id=2197
    b. Search for any *github.com* entries and remove them
5. Using the command line, change directory to the Repositories folder you just created
6. Clone the repository: **git clone https://github.com/ssu-cs370-s19/AndroidLab4.git**
7. Change directory to the AndroidLab4 folder you just cloned
8. Create a new branch in the git repository called **LastnameFirstname**
9. Open Android Studio, and open the project

Part 1: Deserializing JSON into Java Objects

Transforming a JSON object into Java objects involves navigating through the JSON hierarchy, taking the values stored there, and creating new Java objects to store those values.

There are three main types of elements in JSON:

- Primitive type. (String, int, float, boolean, etc.)

```
"lemon"  or  4  or  0.5
```

- JsonObject: a collection of name-value pairs. The value can be any valid JSON element.

```
{
  "sweet": 0.3333333333333333,
  "sour": 0.16666666666666666
}
```

- JsonArray: a list of unnamed values. Again, the value can be any valid JSON element.

```
[
  "cold water",
  "fresh lemon juice",
]
```

1. Look in the model directory, and find the *RecipeModel* class. This Java class is meant to represent a single match from the Yummly database API.
    a. It already has a String variable that will hold the recipe's name
    (note: the variable name matches the JSON key name, but it isn't required to)
    b. Add another variable to hold the recipe's *rating*, with getter and setter.
    c. You may continue to add more variables to match the sample. This lab will only use the name and rating for now.

2. Look in the utility directory, and find the *RecipeParser* class.
    This will contain the logic to deserialize the response JSON into a RecipeModel

3. Open the *RecipeParser* class and examine the *getMatches* function.
    Note that some work is already done – it creates a List that we can store the Java models in, and it creates a JSONObject from the input string.

4. Finish the *recipeFromJson* method by continuing to traverse the JSON tree and fill in a *RecipeModel* with the data.
    a. get the matches (an array), save to a variable
    b. get the first match from the array (an object), save to a variable
    c. get the `recipeName` and `rating` from the first match, save to variables
    d. create a new `RecipeModel`
    e. set the model's name and rating
    f. add the model to the list of models

    ```
    modelList.add(model);
    ```

    For now, we're only interested in the first response in the `matches` array. Ignore the rest.
    Look at the block comment below the class to see an example JSON response.
    (click in comment –alt-enter –inject language – json)

Part 2: Update the UI with data from models

Now that the models are deserialized, the data needs to make it onto the screen.

To send data from the RecipeParser to the Activity, complete the **Listener** pattern:

5. Open *RecipeSearchAsyncTask*. The *doInBackground* method is already defined here, but the RecipeListener interface is incomplete.
    a. Define a method prototype that takes a `List<RecipeModel>` parameter. This method is how we will send the models back to the listener.
    b. In onPostExecute, call the listener's method and pass `recipeModels` to it

To put the data on the screen where the user can see it:

6. Look in *MainActivity*. Add code to assign instances of the layout elements to the corresponding private class variables (using *findViewById*).
7. Add a new ClickListener to the search button:

```
searchButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

    }
});
```

8. Inside the *onClick* method, add code to:

```
// create a new task
RecipeSearchAsyncTask task = new RecipeSearchAsyncTask();
```

```
// create a Listener and add it to the task
// (this is based on the Listener contract you defined in AsyncTask)
task.setRecipeListener(new … );
```

```
// start the task
String searchTerm = searchEditText.getText().toString();
task.execute(searchTerm);
```

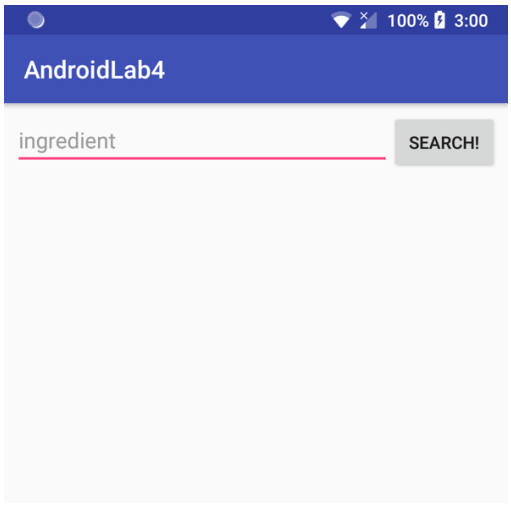
9. In your new RecipeListener instance, add this code to your callback method:

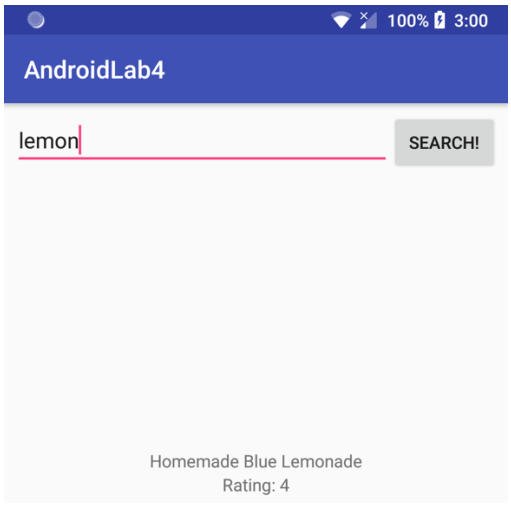
```
// show the first response on the screen
RecipeModel first = models.get(0);

recipeName.setText(first.getRecipeName());
recipeRating.setText("Rating: " + first.getRating());
```

This grabs the first result from the list, and loads its data into TextViews so users can see it.

10. If your app is running correctly, you should be able to enter a search term, click the button, and see the name of the first result appear. If not, revisit the above steps and make sure you've followed all the instructions.





11. When your app runs properly, commit your changes to your branch and push it:

```
git add .
git commit –m "working code complete"
git push origin yourbranchname
```

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

This completes Lab 4.