Spring 2019

Lecture 10: Mobile Application and Product Development

NYC Tech-in-Residence Corps
rdomanski@sbs.nyc.gov

Bhargava Chinthirla
CUNY John Jay College

Eric Spector
CUNY John Jay College

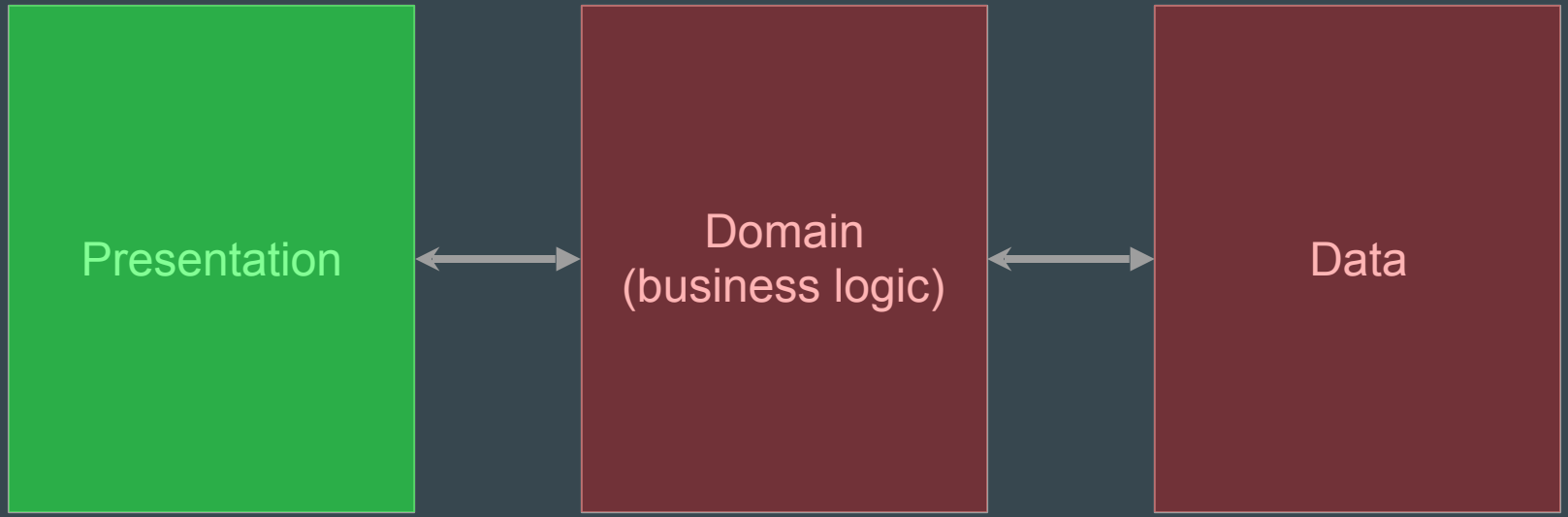
CSCI 380-04

Mobile Application and Product Development

04/03/2019

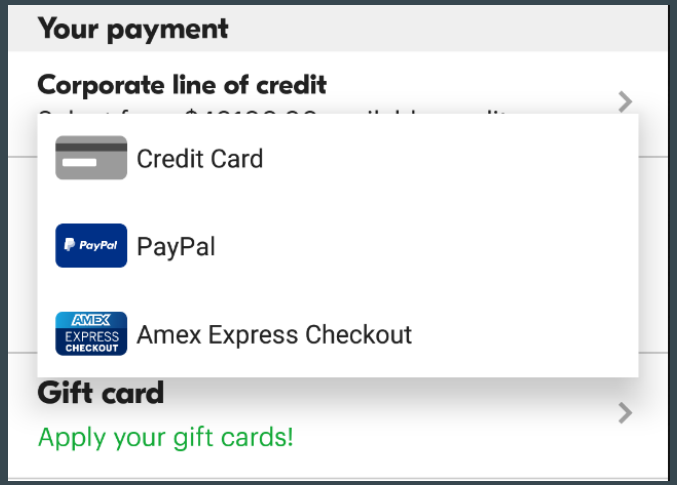
Recap

Mobile application layers



Business logic

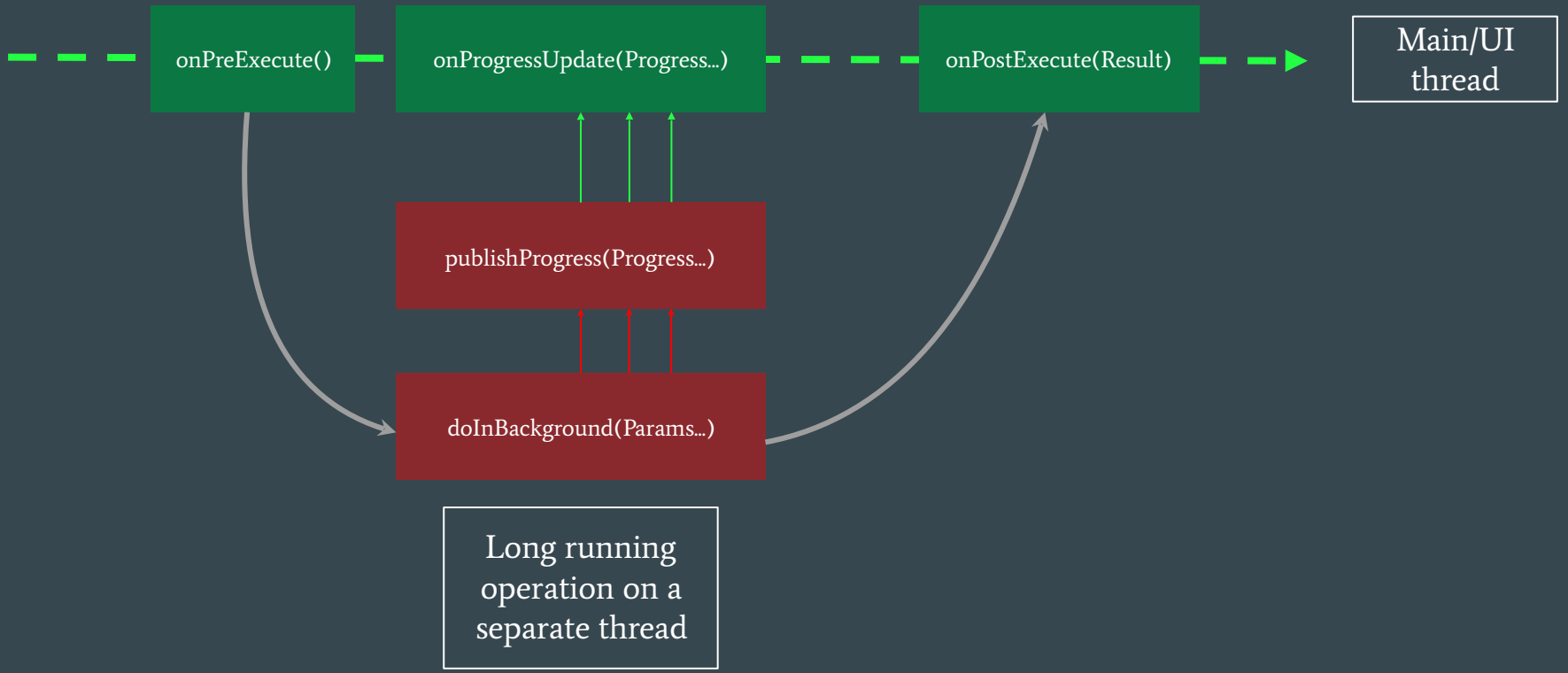
- Also known: business rules, domain logic
- Implements the “real-world” rules of how data can be created and modified
- Real world example: payment methods:



Domain layer

- Prefer doing data manipulation in a background thread
- Data manipulation involves conditionals, loops, mathematical operations, etc.
- This layer is the unique selling point (USP) of your application, the rest is really just “glue code”
- Do not access framework (i.e., android) classes from the domain layer, as it adds extra overhead and can make testing hard

Attached Threading API - AsyncTask



Retrofit: asynchronous calls

- Enqueue a call:

```
ServiceCreator.createSpotifyService()
    .getCategories()
    .enqueue(new Callback<CategoriesResponseModel>() {
        @Override
        public void onResponse(Call<CategoriesResponseModel> call,
                               Response<CategoriesResponseModel> response) {
            if (response.isSuccessful()) {
                CategoryItem categoryItem = CategoryItemUtil.getRandomCategoryItem(response.body());
            }
        }

        @Override
        public void onFailure(Call<CategoriesResponseModel> call, Throwable t) {

        }
    });
```

Retrofit: synchronous calls

- Execute a call:

```
try {
    final Response<CategoriesResponseModel> response = ServiceCreator.createSpotifyService()
            .getCategories()
            .execute();
    if (response.isSuccessful()) {
        CategoryItemUtil.getRandomCategoryItem(response.body());
    }
} catch (IOException e) {

}
```

Your turn - pick up from last time

```
implementation 'com.squareup.retrofit2:retrofit:2.5.0'
implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
```

Dependencies

```
{
    "id": 166889231,
    "full_name": "bhargman/assignment1",
    "private": false,
    "html_url": "https://github.com/bhargman/assignment1",
    "description": "Assignment 1 for CSCI 380-04 - Due by 11:59 PM, Feb 19th, 2019"
}
```

Sample JSON of a GitHub Repo

```
public interface GitHubService {
    @GET("users/{user}/repos")
    Call<ResponseBody> listRepos(@Path("user") String user);
}
```

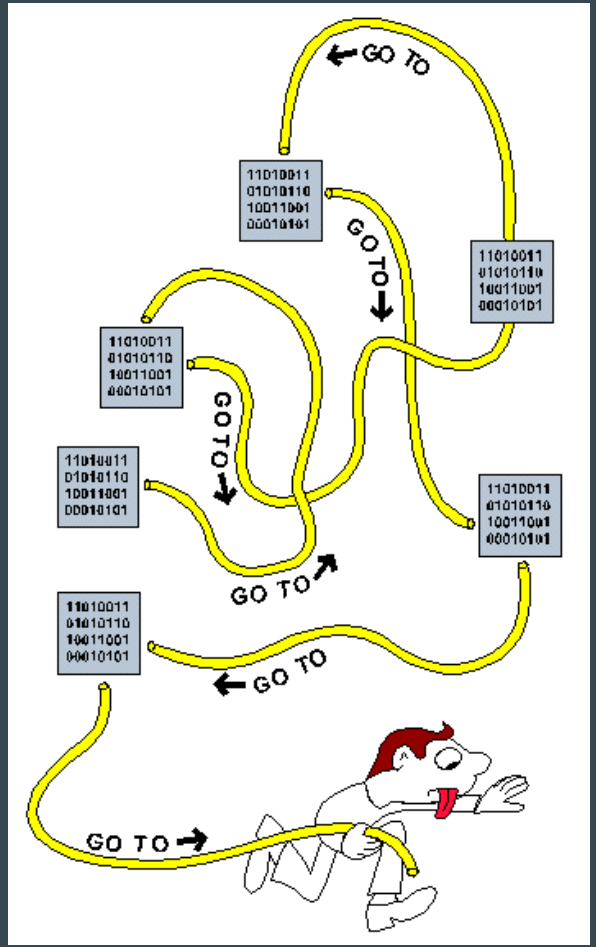
Representing the GET repos API as an interface method

Glue code

- Serves solely to "adapt" different parts of code that would otherwise be incompatible
  - Usually, code that just feels like it's passing through to some other class' code can be seen as glue code
  - An example of this is SharedPreferencesHelper from assignment 3
- We can create a similar class to hold all of our API calls

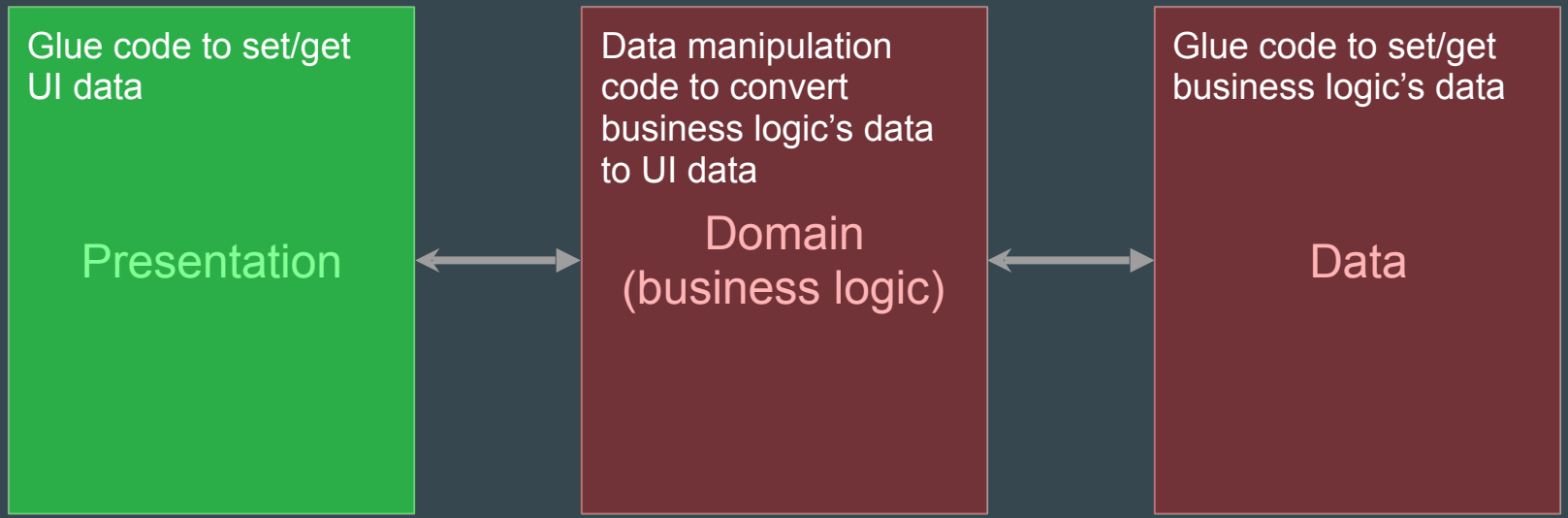
Spaghetti code

- Trying to mix glue code and business logic code all in one giant class can lead to spaghetti code
- Spaghetti code eventually leads to untestable code, and makes your application hard to extend



Separation of concerns

- To avoid spaghetti code and other architectural pitfalls, keep in mind the mobile application layers, and how they separate each of their own concerns

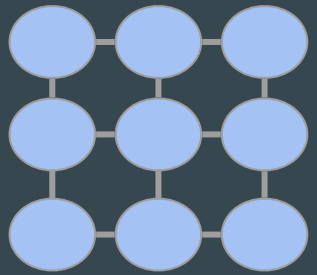


Data structure types

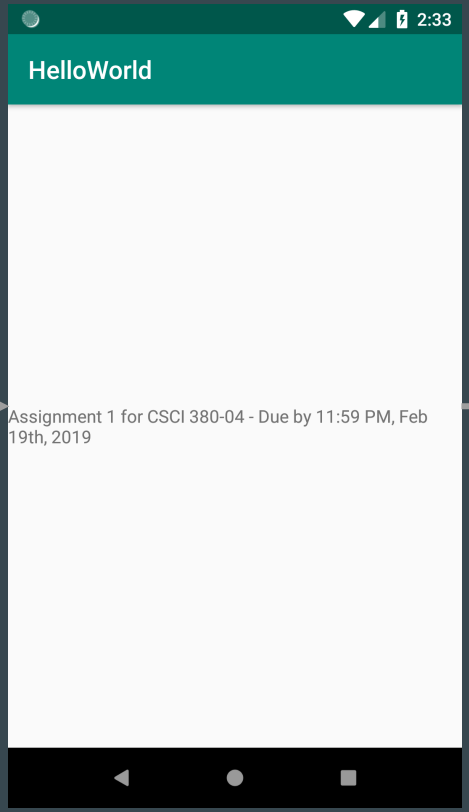
- Scalar: single object (e.g., String, Integer, Boolean)

- Linear/Vector: a list of objects (e.g., List, Set, Queue)

- Non-linear: a multi-dimensional group of objects, not arranged in any specific manner (e.g., Map, Graph, Tree)



We knew we'd only be showing one repo's description, so we only placed one TextView inside of our activity's layout XML



What if we want to show an n number of repo descriptions?

Bad approach

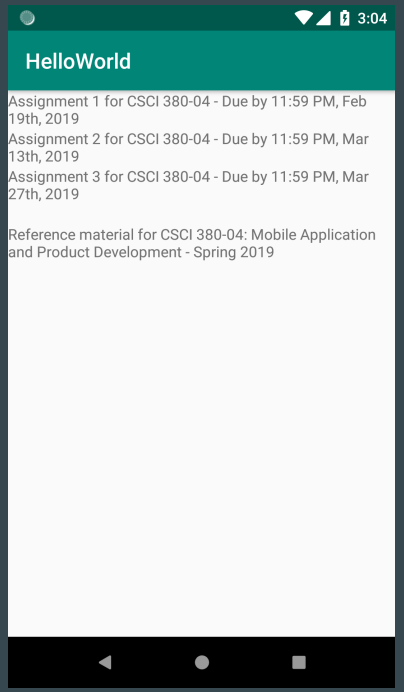
- We can manually create views for each item in a list and add them as children of a layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/layout"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent">

</LinearLayout>

...

this.linearLayout = findViewById(R.id.layout);
for (Repo repo : repos) {
    final TextView textView = new TextView(this);
    textView.setText(repo.getDescription());
    linearLayout.addView(textView);
}
```

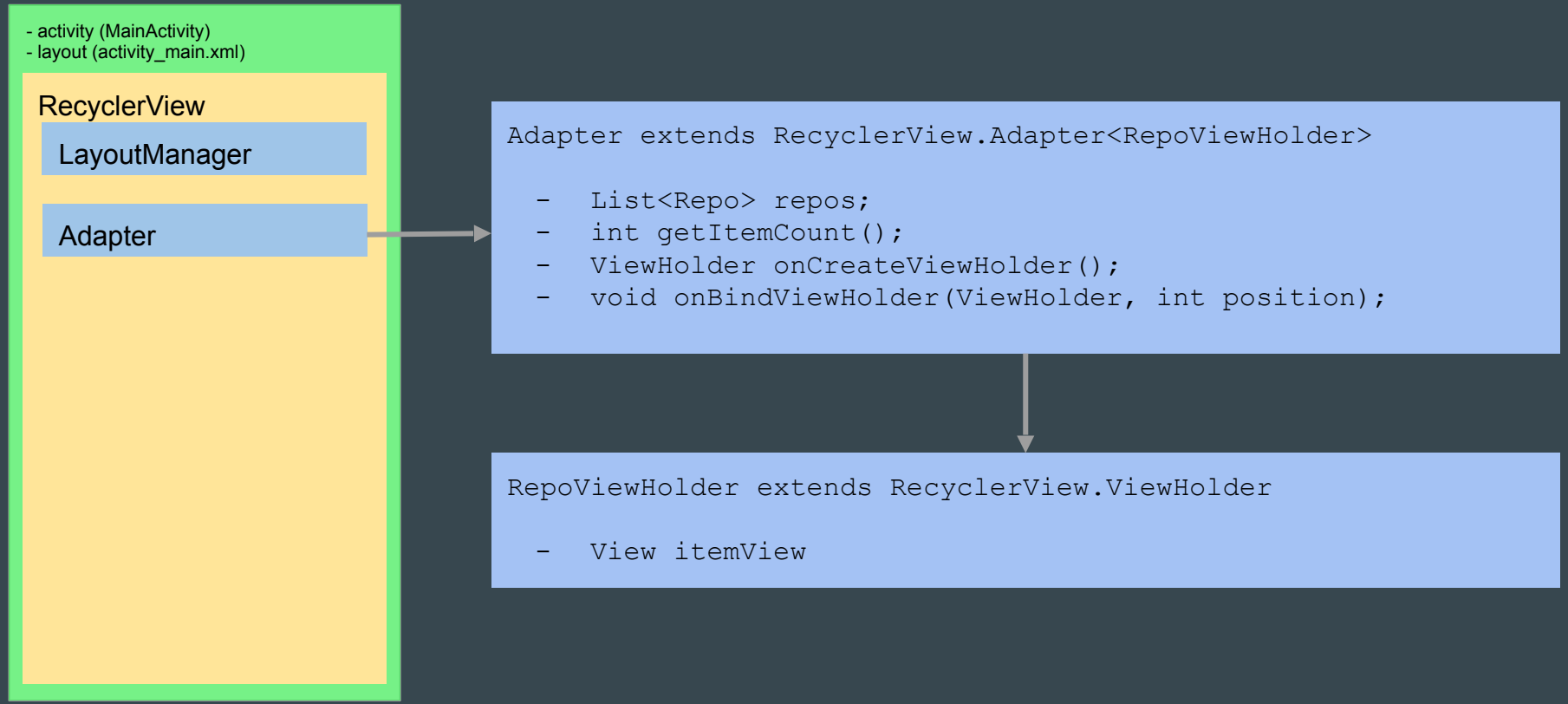


Good approach - RecyclerView

- RecyclerView is useful for displaying a scrolling list of elements, because it scales up to a large value of n and also handles recycling views that go off screen
- In order to recycle views, it uses the View Holder pattern
- It's a more advanced and flexible version of ListView (the older, clunky, error-prone way of showing a list of items)
- Requires a new dependency:

```
implementation 'com.android.support:recyclerview-v7:28.0.0'
```

RecyclerView components



```
class RepoViewHolder extends RecyclerView.ViewHolder {

    final TextView textView;

    RepoViewHolder(@NonNull View itemView) {
        super(itemView);
        textView = itemView.findViewById(R.id.text_view);
    }
}
...
@NonNull
@Override
public RepoViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) {
    // create a view_holder_repo.xml layout file that represents RepoViewHolder!

    final View itemView = LayoutInflater.from(viewGroup.getContext())
        .inflate(R.layout.view_holder_repo, null, false);
    return new RepoViewHolder(itemView);
}

@Override
public void onBindViewHolder(@NonNull RepoViewHolder repoViewHolder, int i) {
    repoViewHolder.textView.setText(repos.get(i).getDescription());
}

@Override
public int getItemCount() {
    return repos == null ? 0 : repos.size();
}
```

Requirements for final project

- Must use some sort of API (similar to how you used Spotify's API for assignment 3)
  - Talk to me now or email me if you can't find a suitable API for your project's idea!
- Must use SharedPreferences in some sort of manner to persist user data
- Make sure to follow the Mobile Application Layers that we've been talking about, it'll help with unit testing the domain layer (we'll talk more about testing next week)
- Testing requirements will be specified at the end of the next lecture