Espresso for Testing

Today

- We’ll cover some testing techniques for Android using the Espresso framework.
- We’ll discuss some popular/common design patterns used in object-oriented programming (C++ and Java).
- We’ll wrap up with a review of everything we’ve done so far.

Espresso for UI Testing

Espresso: Introduction

- Espresso is a framework for UI tests in Android.
- Espresso tests state expectations, interactions, and assertions, but no boilerplate required.
- Espresso is for developers.
- Not really used for black-box testing, but useful if you know what the UI and code ought to be doing.
- Espresso actually waits until AsyncTask is done to test that particular UI element!
  - Some of you have run into issues with debugging AsyncTask.

Espresso: API Components

- 4 main components:
  - Espresso: entry point to interactions with views.
  - ViewMatchers: pass these to onView() method to locate views within current view hierarcy.
  - ViewActions: ViewAction objects can be passed to ViewInteraction.perform(). Eg., click().
  - ViewAssertions: pass ViewAssertion objects to ViewInteraction.check().
- Let’s see some more details.

Espresso: Workflow

- The basic workflow for an Espresso test is as follows:
  - Find a view.
  - Perform an action on the view.
  - Validate an assertion.
- In code, this looks like:
  - onView(ViewMatcher).perform(ViewAction).check(ViewAssertion).
- Example of ViewMatcher is the view id (R.id.stuff).
- Example of ViewAction: click() (simulate a click), typeText() (simulates typing text), pressKey(), and clearText().
- Example of a ViewAssertion: isDisplayed() (check if the element is displayed), matches(), check() etc.

Espresso: Workflow

- You write your Android app as usual.
- Then, you write an Espresso test program.
- You pick over UI elements, write little Espresso code snippets to test them.
- How do you test?
- You think about what user interactions may occur with that UI element, and then simulate it with Espresso.
- Espresso can do more!

Espresso: Workflow

- Espresso-Intents lets you validate and mock Intents.
- You can intercept Intents and provide responses for Activities that are waiting for results.
- Espresso-Contrib lets you test RecyclerView and Navigation Drawer functionality.

Review of Android Programming

Android Review: Activities and Fragments

- Let’s do a quick review of Android programming concepts.
- At the UI level, we have Activities and Fragments.
- Fragments are reusable UI chunks.
- Pass data between Activities using Intents.
- Pass data from Activity to Fragment using setArguments() and getArguments().
- Pass data from Fragment to Activity by implementing a listener/callback using an interface. This is an example of the **Observer** pattern.
- Fragments get their own layout files, typically inflated into a FrameLayout within the Activity layout file.

Android Review: Activities and Fragments

- Activities have lifecycles, and your program needs to be lifecycle aware.
- Override various callbacks to achieve lifecycle awareness.
- Fragment lifecycles are an additional pain on top of this, and they interact with Activity lifecycles.
- An easy way to make Fragments configuration-aware is to call setRetainInstance(true), which will preserve the Fragment instance across Activity recreation.
- Make sure Fragments do not have dependencies on each other. Activity is the traffic controller.

Android Review: RecyclerView

- RecyclerView is an efficient way to show lists of objects on the screen.
- Four components to a RecyclerView:
  - LayoutManager (linear/grid/staggered grid).
  - ViewHolder.
  - Adapter.
  - Animator (we never touched this).
- ViewHolder is defined **inside** the Adapter class.
- The idea is that each ViewHolder is bound to the RecyclerView automatically, as needed.
- We need an XML file for a generic RecyclerView item.

Android Review: Connectivity

- We discussed fetching data from the internet using raw Java code.
- We saw how to do the same task using Request and RequestQueue in the Volley library.
- We also tried downloading files using DownloadManager.
- We explored JobScheduler for downloading files (and in general scheduling tasks) according to some constraints.
- In the context of Volley, we saw the Singleton pattern for RequestQueue (ensuring only one RequestQueue exists for whole program).

Android Review: AsyncTask and AsyncTaskLoader

- To ensure we didn’t hang the UI thread, we downloaded data using AsyncTask.
- This presents challenges in context of lifecycle awareness: zombie threads can be created.
- The old solution to this was to use AsyncTaskLoader, which automatically handled configuration changes.
- Loaders are a deprecated pattern now in Android.
- The replacement to Loaders is a ViewModel.