Lab 04: Android Walkthrough

Prerequisites

This lab is based on an Android project so you will need a Windows or OSX based development environment with the Android tools installed.

Downloads

Download the starting solution and a completed version from http://university.xamarin.com.

Lab Goals

The goal of this lab will be to demonstrate different performance issues you may encounter in an Android project. It is best to run the program on a live device to see most of the issues, as the emulator will either hide them, or encounter them in different ways.

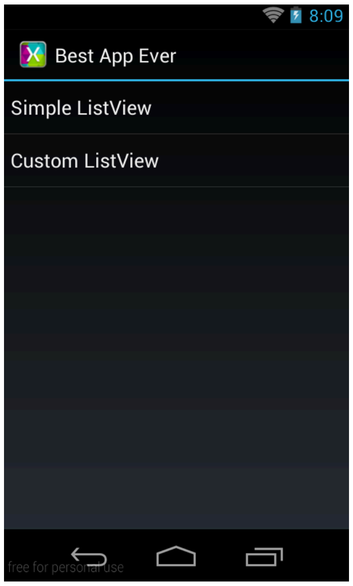
Unlike other labs, this will not walk you through a solution. Instead, it will point out some of the issues with the application and let you attempt to fix them.

If you cannot figure out the problem, or possible solution, there is a `Completed` form of this lab that has fixes with comments labeled as TODO items so you can easily locate the changes. For the best learning experience, we recommend you play with the lab though – it's instructive to see what fixes issues and what doesn't.

Known Issues with "The Best App Ever"

Open the Starting Solution

1. Launch Xamarin Studio and open the starting solution in the begin folder – `Lab4.TheBestAppEver.sln`.
2. Run the program – preferably on a real Android device. It will show a list view with two choices:



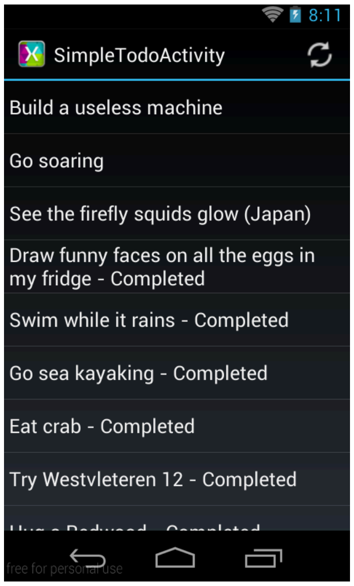
3. Each corresponds to a specific activity which illustrates some common performance issues you will encounter in Android.

Simple ListView (TODO: Step 1)

1. The Simple ListView activity is using a built-in ArrayAdapter<T> to display a set of strings. It should appear to run just fine with the limited amount of data it has. Go ahead and tap on it to see the data.

Note: if you are using the Android SDK emulator (vs. Genymotion or a device), you may have trouble executing this activity. This is because the emulator has very low limits on the number of strong-referenced objects allowed and it begins to garbage collect constantly when you hit that limit. We recommend running on a real device for this lab.

2. It displays a list of bucket list items to complete in your life. Scroll the list so see how performant it is on your device.



8:11

SimpleTodoActivity

Build a useless machine

Go soaring

See the firefly squids glow (Japan)

Draw funny faces on all the eggs in my fridge - Completed

Swim while it rains - Completed

Go sea kayaking - Completed

Eat crab - Completed

Try Westvleteren 12 - Completed

4. If you look in the Output Window you should see that the total time to load the ListView with the data. This is essentially how long it took to create the adapter and present the view with data, and doesn't really include the entire layout and render time, but we can use it across the board to get a sense of performance. Alternatively, try using the Hierarchy Viewer, which is part of the Android Debug Monitor tool (monitor in the Android tools).

   Go ahead and write down the number for comparison. This is for 2000 items.

5. Stop the program and change the data to include a larger data set. Open the SimpleTodoActivity and find the OnRefresh method and adjust the value passed to the GetItems method as shown below, bump it to 50,000.

```
async Task OnRefresh() {

    ListAdapter = null;

    var todos = (await WebService.GetItems(this, 50000));

    using (var t = new Timer("SimpleTodoActivity"))
    {
        ListAdapter = new ArrayAdapter<TodoItem>(this,
           Android.Resource.Layout.SimpleListItem1,
           todos.ToArray());
    }
}
```

6. Run the app again and select the SimpleTodoActivity. It should be *very* slow, if you look in the output window, you will find a slew of GC messages indicating that there are too many outstanding GREFs (global references) indicating that the Dalvik GC is working too hard. It likely won't return in a timely fashion, so go ahead and close the app.

   What you are seeing here is a common issue when using the ArrayAdapter. It converts all your objects (TodoItems) into Java objects to store in a Java array.

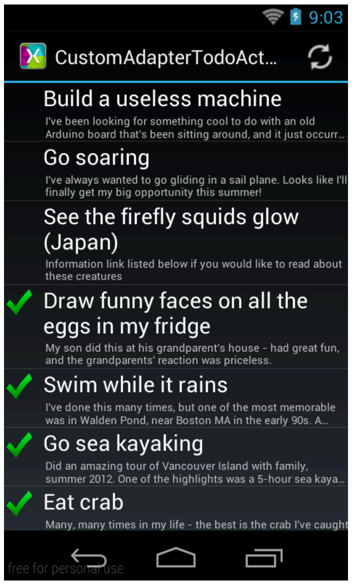
That causes more work for the Dalvik virtual machine to collect – and at some point we just overwhelm it. The solution is to create a custom adapter – something you should generally always do.

7. Stop the program and change the ArrayAdapter into an EnumerableAdapter<T> - this is a custom class included in the Adapters folder, which has been pre-built for you. Run the app again and note the time now for 50k items. You should be able to scale up to hundreds of thousands of items with this approach, however it can be improved.
8. Look at the EnumerableAdapter implementation – particularly the GetView method. Are there any improvements you can make here?

Hint: think about view recycling – remember that's built into Android.

Custom ListView (TODO: Step 2)

1. The Custom ListView activity is using a custom adapter with a custom view representation to show check marks. to display a set of strings. It should appear to run just fine with the limited amount of data it has. Go ahead and tap on it to see the data.



Build a useless machine
I've been looking for something cool to do with an old Arduino board that's been sitting around, and it just occurr...

Go soaring
I've always wanted to go gliding in a sail plane. Looks like I'll finally get my big opportunity this summer!

See the firefly squids glow (Japan)
Information link listed below if you would like to read about these creatures

✓ Draw funny faces on all the eggs in my fridge
My son did this at his grandparent's house - had great fun, and the grandparents' reaction was priceless.

✓ Swim while it rains
I've done this many times, but one of the most memorable was in Walden Pond, near Boston MA in the early 90s. A...

✓ Go sea kayaking
Did an amazing tour of Vancouver Island with family, summer 2012. One of the highlights was a 5-hour sea kaya...

✓ Eat crab
Many, many times in my life - the best is the crab I've caught

2. It has a lot more data being added to the screen, but also has an inefficient implementation for the adapter. There are several things you can improve:
   a. Use view recycling, just as you did before.
   b. Look at the view layout itself. Is there a way to reduce the nesting or cut down on the visuals being created?
   c. Look at the images being added – is there a way to improve it? In particular, compare the size of the image itself with the size we are using in the display.

   d. Think about caching off the view elements in the custom layout somehow to avoid the FindViewById<> calls each time. Also, is there a way to improve how the image is displayed?

3. As you make changes, verify them by checking how long it takes to load – also try using Hierarchy Viewer and ADM to verify the memory usage and visual layout times.

Summary

Congratulations on working through some real world (although simple) code which has performance issues! You can look at the pre-canned solution in the completed project but keep in mind that there are multiple ways to solve the issues in this project so the completed version isn't a definitive solution – just an example way to solve some of the issues we have.

The completed solution has a more finished version of the EnumerableAdapter<T> as well – where you can customize the text, detail and item id of the adapter through property delegates.