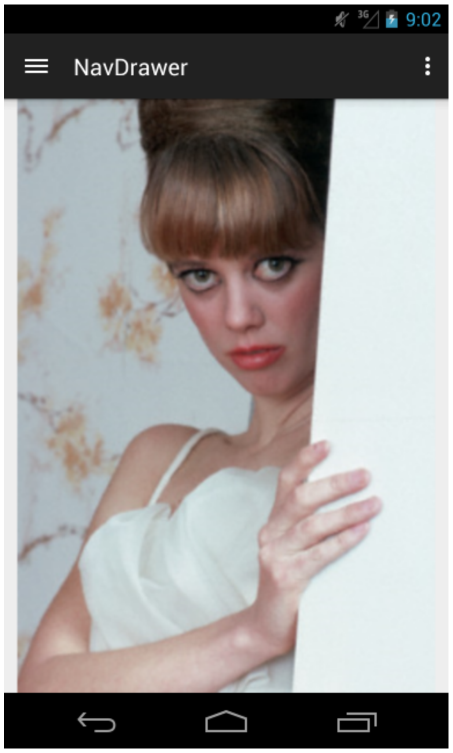
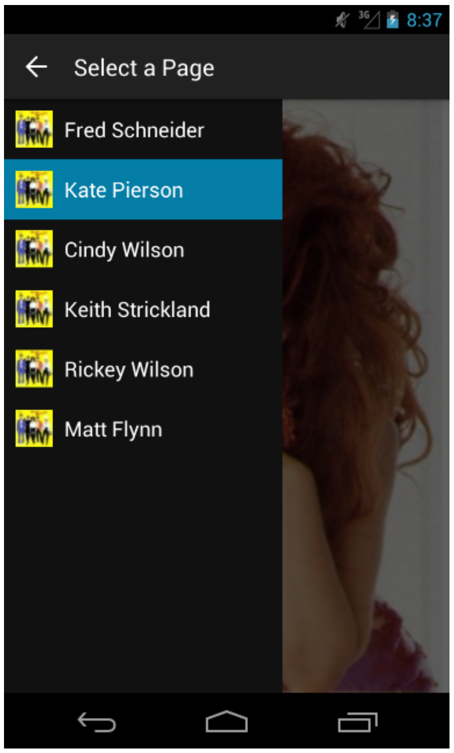
CSCD 372: Android Development
Lab 2: Navigation Drawers (Toolbar Version)

Lab Overview

This lab covers the use of Navigation Drawers. Like other list-based views in Android, this requires an adapter class. In this case we won't have to provide our own adapter, but will instead illustrate some expanded capabilities of ArrayAdapter. Navigation Drawers also require some additional work in order to make them interact with Toolbars (or ActionBars). We'll keep it as simple as possible by using the Nav drawer to select photos. I chose to use photos of band members from the B-52s, one of the boppin'est bands of the 80's and 90's, and still touring. Two of these are former members: Rickey Wilson is deceased, and Matt Flynn was a traveling member and currently with Maroon 5. Feel free to provide your own photo collection of some kind for this lab, but you'll want to keep them smallish for the emulator (mine are in the neighborhood of 240 x 320). This is a short lab, but don't procrastinate, as the next one is longer. Let me know if you want to see an ActionBar version of this lab (it is slightly different).





Following are suggestions and hints for how to proceed.

Task 1: Create a project shell and layouts

1. The code generated by the Navigation Drawer wizard is overly complex for our purposes here (I'll walk through it later if time permits), so use the Blank Activity template instead. Please use the standard naming convention for Lab projects. You can immediately remove the pink mail button from activity_main.xml and the associated code in onCreate().

2. If you want to use my images, download B52s.zip from the course website. Copy them to your drawable folder. Normally you would want to provide multiple resolutions for the small logo image, but we won't fuss over such details for an exploratory lab.

3. Change your content_main layout to follow the navigation drawer template layout that you can find here, under the heading "Create a Drawer Layout":

   http://developer.android.com/training/implementing-navigation/nav-drawer.html

   The easiest way is to just copy and paste that in text mode for the layout, except that you want to retain the app:layout_behavior=(...) and the xmnls:app=(...) attributes from the originally generated content_main.xml in the header for DrawerLayout. The FrameLayout section is for your main view content (often a Fragment, but not in this case), and the ListView portion is for the Navigation Drawer. The encapsulating DrawerLayout allows the ListView to overlay the FrameLayout when the Drawer is slid out.

4. Add an ImageView widget to the FrameLayout section. I did this in Design View. Set the width and height to match_parent and initialize the source to one of your images.

5. In your activity, it'll be convenient (although not absolutely necessary) to have member variables for the DrawerLayout and the ListView. Considering adding those, and initialize them from the layout that is inflated in onCreate().

6. Add a layout resource for a row of the navigation drawer. Call it something like "nav_list_row.xml" (right-click on your layout folder). RelativeLayout is probably easiest here. The width should be match_parent and the height should wrap_content. Drop an ImageView on the left for the icon and a TextView widget next to it for the photo name. Set the src for the icon to the logo image. You'll probably want to specify a light color for the text since the default Nav Drawer background is dark (at least at the time of this writing). Setting a dark background for the TextView will then make it easier to see in the preview, but isn't strictly necessary.

Task 2: Add an Array of Names and an ArrayAdapter for the ListView

1. You're going to need an array of strings for the photo names. Do that as an array resource in your strings.xml file.

2. In onCreate(), instantiate an instance of ArrayAdapter. Start by reading the short Class Overview in the API docs in order to determine which constructor you want. All of them require a resource argument, which is either the id of a TextView or the id of a layout that contains a TextView. In our case it is the latter. Because of that, we also need to give it the id of a TextView within that layout.

3. You'll also need to pass the array of string names. To do this you'll need to convert your array resource to a Java array, which you can do by calling getResources() and getStringArray(). Note that this also determines how you need to set the generic for the ArrayAdapter class.

4. Do a test run now. You should be able to slide the Nav Drawer out from the left and slide it back. Fix up anything that looks bad in the layout.

5. Note (from the ArrayAdapter Class Overview) that we could also set a unique image in the drawer for each name, but in that case we'd need to provide an extension of ArrayAdapter that overrides getView(). We take that approach in another lab, for a more complex ExpandableListView.

Task 3: Handle selection of Nav Drawer rows

1. Have your Activity implement ListView.OnItemClickListener, and make the call to register as a listener in onCreate().
2. Add the required onItemClick() method. Respond to the click by changing the source for your ImageView object in the main layout. A local array of integers to hold the drawable resource IDs is helpful here (as opposed to a big case statement). You can hard-code the resource id's or load them into a java array using a typed-array, as shown in the lecture notes (module 4).
3. Do a test run. You should now be able to change the main display by clicking a Nav Drawer row. Don't celebrate yet. You still need to make the Nav Drawer interact properly with the Toolbar. This means getting a dynamic icon up there that can be used to slide it, getting the Title to change when it is out, and removing any inappropriate menu actions while it is out. Fortunately, Android makes this fairly easy by providing the ActionBarDrawerToggle class (which works with either ActionBars or Toolbars).

Task 4: Add ActionBarDrawerToggle

1. Add a couple of final Strings to hold two different titles for the Toolbar. One will be displayed when the drawer is hidden and the other when the drawer is exposed. For the former you can grab the default title (generated from your project name) by calling getTitle(). Set the title for an exposed nav drawer to something else. As seen above, I chose "Select a Page".
2. Also in onCreate(), create an instance of ActionBarDrawerToggle. The constructor requires several arguments, which are documented (almost correctly) under the heading "Listen for Open and Close Events" here:

   http://developer.android.com/training/implementing-navigation/nav-drawer.html#OpenClose

   That page assumes you'll be using the version from android.support.v4.app, which your compiler and the API will tell you is deprecated. What you want is the version from android.support.v7.app, so make sure you use the proper import. You'll have to open that documentation (search for support-v7-appcompat.ActionBarDrawerToggle) to find the correct argument list for the constructor (use the version that takes a reference to your Toolbar). The content description resources are used to describe the drawer state using text to speech for visually impaired users, so set them to meaningful strings.

   Follow the example for instantiating the ActionBarDrawerToggle and registering it as a listener on the DrawerLayout. In order to avoid an exception, change getActionBar() to getSupportActionBar(), and note that you can make a direct call to setTitle(), without first getting the action bar. The call those handlers make to invalidateOptionsMenu() has no effect yet, but will become important later.
3. After setting the toggle as a listener, add this call to activate a navigation icon on the Toolbar:

   ```
   getSupportActionBar().setDisplayHomeAsUpEnabled(true) ;
   ```

   Then call this to get it to change its appearance in synchronization with the Nav Drawer:

   ```
   mDrawerToggle.syncState() ;
   ```
4. Do a test run and make sure you can actuate the drawer with the icon. The animation is a bit of a cheap thrill, huh? Hey, we have to grab our fun where we can. We're almost done.

Task 5: Patch up any menu options

1. It may be the case that some of your Toolbar actions or menu options are inappropriate when the navigation drawer is out. Let's remove the default Settings option from the menu when the drawer is out. If you're in the habit of replacing that option with an About option, then remove your About option when the drawer is out. This is where those calls to invalidateOptionsMenu() in the slide handlers come in. Add an override of onPrepareOptionsMenu(). This method is always called right before the menu is shown, and so provides an opportunity to modify it. Make a call on the DrawerLayout to ask it if the drawer is open. If it is, find the Settings (or About) item on the menu and set its visibility to false. The requirement here is that the menu be modified somehow when the drawer is out.
2. Do a test run to ensure that the menu changes when the drawer is out. You might notice on my screen captures above that my menu disappeared altogether. That's because I had only one action on my menu, and removing it makes the menu disappear altogether.

Task 6: Handle rotations

1. Try opening the drawer, selecting a different image, and closing the drawer. Then rotate your display. It would be nice if it retained the last selected image, so let's fix that. What we want to do is somehow save the current image to the outgoing bundle in onSaveInstanceState(), and then restore it in onCreate() when an incoming bundle is present.
2. Hopefully you already figured out how to set that image in Task 3 above. The same approach can be used here, but you'll need to know what resource id was last used in order to set that ImageView. A clever way to do this is to attach a Tag to the ImageView, identifying the resource id that it is currently displaying. So go back to your onItemClick() handler and make a call to setTag(), passing it the drawable resource id that you just gave to the ImageView.
3. In onSaveInstanceState(), you can now make a call to getTag() on the ImageView in order to save that resource id to the outgoing bundle.
4. In onCreate(), when the bundle is present, you can retrieve that resource id from the bundle and use it to set the image just like you did in your onItemClick() handler.
5. Make another test run. Select a different image and rotate. That different image should still be displayed. Then, without changing the image, rotate back. If it crashes, it's probably because you forgot to do something in step 4. Take a closer look at your onItemClick() handler and think about what just happened.

Comments

1. Later on you'll learn about Fragments and how to change the currently displayed Fragment, which you could then combine with your knowledge of Nav drawers. There are also other navigation patterns to choose from (tabs, swiping, swiping with tabs, and using a spinner).
2. Submit your apk file and the source file for your main activity using the required naming convention, e.g., pschimpfLab2.zip containing pschimpfLab2.apk (or app-debug.apk) and MainActivity.java.