

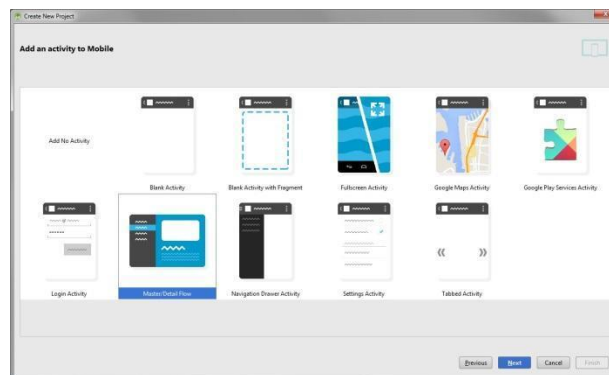
## NOTE:

This modification of activity 2.1.4 has not been approved by PLTW at this time.  
This modification was for editing and clarification only.  
To see the most current version, please refer to [www.PLTW.org](http://www.PLTW.org)

## Activity 2.1.4 Navigation Drawer Basics

### Introduction

There are many different design patterns for getting around within an app: Master-detail view, Navigation Drawer, and Popover Menu, just to name a few. You've probably seen these and others in apps that you've used. The **File > New Project...** menu option in Android™ Studio will provide some default navigation types. The College App was created using the Navigation Drawer template.



**Add an Activity to Mobile**

In this activity, you will learn how to modify a navigation drawer created in this fashion to suit your app's needs.

### Materials

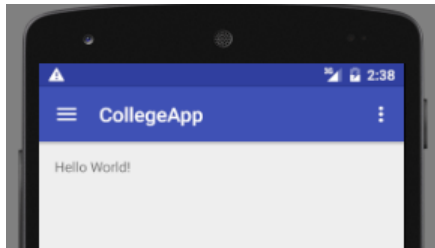
- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator

### Activity

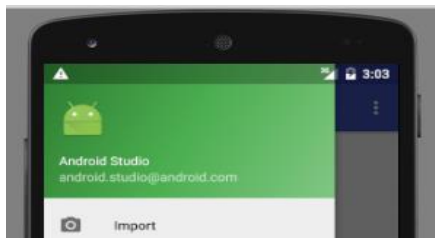
1. Form pairs as directed by your instructor.
2. Greet your partner and set team norms for pair programming.

## Part I: The Nav Drawer Template

3. Create a new project in Android Studio. Give this project the following values (when no value is specified, leave the defaults):
  - a. Application name: **CollegeApp**
  - b. Company Domain: **examples.pltw.org**
  - c. Store your project in your *AndroidProjects* folder.
  - d. Minimum SDK: **API 22**
  - e. Add an Activity to Mobile: **Navigation Drawer Activity**
  - f. Activity Name: **ApplicantActivity**
4. Run your app. You should see a screen like the one shown below. Do not confuse this with the layout version you currently see on the studio screen.



5. Click the hamburger menu (three horizontal lines ≡) to bring out the navigation drawer shown below.




6. Previously you learned about the basic XML files found in Android Studio. When you use navigation drawers, more XML files are needed. These files can be found in the *layout* and *menu* project folders. They are described below.
  - *ApplicantActivity.java* contains presenter layer code for the app.
  - *activity\_applicant.xml* contains the overall layout of the main activity.

- *app\_bar\_applicant.xml* contains layout information for the blue bar at the top of the screen and the pink floating action button circle.
- *content\_applicant.xml* contains the layout information for the main section of the screen displaying “Hello World!” with a light grey background.
- *nav\_header\_applicant.xml* contains the layout information for the green header area of the Navigation Drawer, which has the Android icon in it.
- *activity\_applicant\_drawer.xml* contains the layout information for the menu items found in the navigation drawer, like icons and ID tags.
- *applicant.xml* contains the layout information for the menu that appears when a user taps the icon to the far right of the blue action bar.

7. When you are creating menus, you have 3 components: Icons, ID tags, and a connection to fragment pages.

The Icons are found in the same place as the ID menu tags. Look at the list above to determine which xml file you should open up.

8. In the xml file you just opened, click on the text tab and take a look at how it is laid out. The upper section is put together with a group while the lower section is a menu list with an item title “Communicate”.
  - We use groups when we want a choice between multiple options.
  - We use menus when we want to create submenus, or menus within menus.Remove the sub-menu titled Communicate from the app by deleting the XML item element and menu.
  - Try to run your app now.
9. What errors do you get? These errors are because you have not removed the java code that links those menu items to the program. Delete the lines in your Java code in *ApplicantActivity.java* that are producing these errors.
10. Run your app and test it to see what has changed.
11. Summarize in your own words the steps it takes to remove a menu item from the Navigation Drawer.

12. The next step is to change the remaining menu options to something that makes sense for our program. To do this, we will need to change the icon and the id. Before you run anything, complete steps 13 and 14 as they go together.
13. In your xml file, change the menu item, currently called “Import”, so that it has the following attributes:
  - Id: `@+id/family_member`
  - Icon: `@drawable/ic_group_black_24dp`
  - Title: `Family Member`
14. The `ic_group_black_24dp`, along with many other **material design icons**, can be included in your project by using Vector Asset Studio. Follow these steps:
  - a. Right-click on your `res` directory.
  - b. Select **New > Vector Asset**.
  - c. Make sure that **Material Icon** is selected.
  - d. Click the **Choose** button.
  - e. Select the **Social** category.
  - f. Select the first icon that looks like: 
  - g. Click **Next**.
  - h. Keep the defaults and click **Finish**.
15. Back in your java file, modify the `onNavigationItemSelected` method to address the id change you just made from `nav_camera` to `family_member`.
16. Repeat steps 13-15 to add the following menu piece:

Vector asset:

`ic_person_black_24dp`

ID tag:

```
<item
    android:id="@+id/profile"
    android:icon="@drawable/ic_person_black_24dp"
    android:title="Profile" />
```

Java Code:

`nav_gallery` to `profile`.

17. Remove the remaining else if statements from your Java and the remaining items from your xml file.
18. Test your app.
19. Remove the floating action button.

To do this, remember that you have to remove it from 2 places, one from Java, the other from XML. For now we're just going to block comment these sections out.

To comment out a section in XML, you would use the following comment tags `<!-- -->` whereas in Java you would still use `/* */`.

In Android Studio, there is a trick to this where you can highlight the contents you want to block then click on Code followed by Toggle block comment.

*Highlight material > Code > Comment with Block Comment*

20. Test your app to verify that it still runs and that the floating action button is gone.

## Part II: Adding Fragments

Navigation Drawer apps use fragments. These allow for smooth modular designs that make it easy to port to tablets or other devices at a later time.

To utilize fragments, you will need to create a home for the Fragment in your layout, as well as the fragment classes for each main class we would like to display. And finally, in `ApplicantActivity`, we would need to write code to give your app logic for how to switch between them.

21. Start by editing your `content_applicant.xml` file. Remove the existing `TextView` element in this file and replace it with a `FrameLayout` element. Give this element the following attributes:
  - `layout_width: match_parent`
  - `layout_height: match_parent`
  - `id: @+id/content_frame`

22. In the `onNavigation` method found in `ApplicantActivity`, create a local variable named `contentFragment` of type `Fragment`, and set its initial value to `null`. (You will have to import `Fragment` if this is the first time using it).
23. Create just one of the Fragments that you will need, the `FamilyMemberFragment`, by creating a new class with that name that `extends` `android.support.v4.app.Fragment`.
- This Fragment will need its own layout. For now, keep it simple.
  - To do this, right click on layout and add a new resource file.
  - Name it `fragment_family_member.xml` and change the root element to `LinearLayout`.
  - Inside the design, create a single `TextView` showing the string literal "Family Member Content".
24. Within `FamilyMemberFragment`, create an override (`@override`) for a public `View` method `onCreateView(LayoutInflater, ViewGroup, Bundle)`.
- Note that each one of those words within the parenthesis are variable typeDefs and will need variables to be passed on as well.*
25. In the first line of this method, call its super class's `onCreateView` method passing in the same parameters that you just created.
26. On the next line, declare a `View` variable named `rootView` and assign it the value retrieved by calling the `inflate` method of your `LayoutInflater` parameter with the arguments `R.layout.fragment_family_member`, your `ViewGroup` parameter, and the value `false` passed within.
- This is how your Fragment will know to look for fragment\_family\_member.xml to get its layout. To get other layout elements later, you will call the `findViewById` method of `rootView`.*
27. At the end of `onCreateView`, return `rootView`. This is as minimal a Fragment as you can create.
28. In your `ApplicantActivity.java`, return to the `onNavigation` method and find the conditional that detects selection of the Family Member menu item.

Inside the braces, assign a new `FamilyMemberFragment` to `contentFragment`.

29. Repeat Steps 23–28, this time to create `ProfileFragment` and a layout file for it similar to the layout file that you created for `FamilyMemberFragment`.

30. After your conditionals, within `onNavigationItemSelected`, type the following:

```
1  if (contentFragment != null) {  
2      FragmentTransaction ft = getSupportFragmentManager().beginTransaction();  
3      ft.replace(R.id.content_frame, contentFragment);  
4      ft.commit();  
5  }
```

31. Test your app to make sure that the Fragments are properly replacing one another.

## Conclusion

1. Which layers of the MVP pattern are currently present in the College App?
2. Of the layers that you listed in the previous question, what parts of the College App belong to which layers?
3. **[U2.03\*]** How did you use inheritance in the College App?
4. **[K3.03]** Would you prefer to use the navigation drawer implementation provided by Google or create your own? Why?
5. **[S6.13\*]** Use the Internet or any other resource to help you find information about three other forms of navigation used within Android apps. Discuss the pros and cons of using each as a replacement for the navigation drawer in the College App.