1. Topic 1: Navigation drawer. This relates to my final project in that I needed to implement one in order to navigate to each of the main three activities in the manner that I desired. I implemented this around the main (budgeting), investment mapping, and settings activities.
   1. URIs
      1. <https://developer.android.com/training/implementing-navigation/nav-drawer.html>
      2. <https://code.tutsplus.com/tutorials/how-to-code-a-navigation-drawer-in-an-android-app--cms-30263>
      3. <http://www.technotalkative.com/part-4-playing-with-navigationview/>
   2. The navigation drawer that appears in so many modern apps, from the left side, is initially implemented by making a DrawerLayout the root tag of the XML style documents and placing everything that is normally a part of the activity inside it, followed by a NavigationView which pertains to the elements that are actually displayed in the drawer. In a separate XML file (in /res/menu/, conventionally), make a menu layout for the items that are to be displayed, and have the NavigationView refer to that file.

Given that the XML for the drawer layout and the navigation view are well-formed and ready to go, the next steps that need to be done in order to get the drawer working like the user thinks it would is to go to the Java class for the activity that is to have the drawer on it. In that Java class, one must include the drawer layout itself as well as a handler for when it is toggled on or off (when the navigation panel is visible or not visible, respectively). There should also be a method within the Java class that handles when menu items are clicked.

* 1. One of the most important points is the ever-present theme of separating the activity’s functionality from its visual design. Of course, the end result can be achieved without doing this, but the separation allows the final product of the panel to be realized in a much more understandable manner.

One of the other most important points is that there should be a separate file for the actual menu items that appear in the drawer. This shows that the drawer is just as much of a menu as is the menu that appears in many Android apps by pressing the three dots in the top right corner of the screen, and it helps the developer to realize that the menu can be customized in one place, and then dropped in as the app:menu=“@menu/(menu\_name)” tag in the navigation view: this is crucial to making the navigation drawer consistent with respect to each activity. If the menu was created manually for each activity that needed a navigation drawer, updating it would be an absolute nightmare.

1. Topic 2: Dynamically adding views to/removing views from the layout. This relates to my final project because I need to do this in several places, and without doing this the app would either be so extremely basic in its functionality or so extremely complex in its code. I do this in the “money out” activity (relating to the loan views) and in the main activity, by adding and removing views that show each week’s transactions, and by adding and removing the transactions for each week.
   1. URIs
      1. <https://www.myandroidsolutions.com/2013/02/10/android-add-views-into-view-dynamically/#.WpSX0edG2Uk>
      2. <https://www.androidtutorialpoint.com/basics/dynamically-add-and-remove-views-in-android/>
      3. <http://android-er.blogspot.com/2015/12/add-and-remove-view-dynamically-with.html>
   2. Sometimes, when creating an Android application, it works to make every view and activity and dialog appear and disappear without adding any other views or information. In most cases, though, this won’t cut it: views need to be added dynamically according to user data and user input. One way to do this is with an instance of the LayoutInflater class. With the LayoutInflater, one can push layout files from their generalized file into another (given) layout, and then the new layout is there for use.

Removing layouts is done pretty similarly; whatever is to be clicked on to remove a view requires an onClickListener that uses the .removeView(nameOfView) method of LayoutInflater instances. This is useful because it allows for so much more flexibility in making layouts; if a user were to want to add something in to view, the LayoutInflater class makes it possible (in one way).

* 1. The most important idea from this topic is that dynamically adding views is ideal for flexibility of layout and user experience. This is because it allows the user to be able to see what they have provided (in terms of data/input of some sort) in a familiar setting (as opposed to raw output from a terminal), while still (more or less) maintaining the sanity of the developer by not requiring them to make a specific layout/view/activity for each of the practically infinite possibilities that the user could come up with.

1. Topic 3: Saving user input/data to a database. I have not implemented this yet, though I have seen it done. I plan to use it in connection with the budgeting activity, storing each “Week” object instance as an entry in a single database for the app.
   1. URIs
      1. <https://www.youtube.com/watch?v=Jcmp09LkU-I>
      2. <https://guides.codepath.com/android/Persisting-Data-to-the-Device>
      3. <https://developer.android.com/training/data-storage/sqlite.html>
   2. Using SQLite within an Android activity class, one can store user data (by inputting data object instances to a database defined by you/me/the developers) for future use, only within the app. This is useful because it allows for returning to the same activity (visually) after having closed the app as well as giving the developer a large amount of power over the use of the information.

Using SQLite in an app is not too complex, though a little bit meticulous. The first step is to extend the SQLiteOpenHelper class in another class definition (separate from the activity where the database is to be used). Using an instance of this class and the .execSQL(“query string”) method of SQLiteDatabase object instances is how the database is interacted with by the rest of the Java code.

* 1. One of the most important thoughts to keep in mind when handling user data with SQLite in Android apps is that the user’s input should be placed into the database at the minimum in the onStop method of the main activity, and it should be accessed (if it is a part of what is viewed, like in my app) somewhere in one of the methods that is called when the app starts (onCreate(), onStart(), onResume()). It is crucial to do this automatically in one of the activity’s pre-defined methods and not by, for example, by the press of a button from a user, because the Android design principles call for developers to care for the user. Automatically saving their place (when it comes to their information) so that they don’t have to worry about doing it or forgetting about it is just one way that we can care for the users and have their data kept on their device, ready for them to be able to access it whenever they open the application.