**Phase 2**

The functionality of my app, which I am calling “Basic Financial,” is not exactly cut-and-dry, though it is certainly not ambiguous what I need to do in order to get it working the way I’d like it.

The main activity that will be opened on a fresh launch of the app will be the budgeting activity. The buttons at the top labeled “prev” and “next” (surrounding “Week”) will change which week is to be displayed (the budgeting tools work on a week-by-week basis). They should have onClickListeners whose onClick methods execute a change in week that is viewed. Because there are eventually going to be an arbitrary number of weeks that can be cycled between, each instance of the main budgeting activity should have an ArrayList of Week objects. The buttons at the top (for changing the current week) therefore just change the index in the ArrayList that is to be displayed. When viewing the first Week, pressing “prev” will not do anything, and when viewing the one with the highest index, a new Week is added to the array list.

Each week (below the prev/next buttons and the text that says “Week”) will display all transactions of that specific week. Each Week will need an ArrayList of Transaction objects due to having a dynamic amount of them. Each transaction should say whether or not it adds money (boolean field), the amount of money that it changes the week by (double field (not float nor int, for use in mathematical methods later)), a label (string field), a possible “special” field (string field, one of “Paycheck,” “Bill,” or “Loan”), and a possible tag that the user can generate themselves (string field). Each transaction will display all of these fields in a row, along with an “x” button to the right of the fields, to delete that transaction. This delete button should do no more than call the .remove() method of ArrayLists on the list of transactions.

Below the transactions there will be a “Total remaining” text area with a number below it. That number will be calculated as a method of the week class by adding the transactions together (adding if it adds money, subtracting if it is a purchase). Below that, the “$20 purchases remaining” number (that appears below it) is calculated by dividing the total remaining by 20 and not including a dollar sign. Below that, the “Max average price of 10 purchases” number is calculated by dividing the “Total remaining” number by 10 and including a dollar sign.

The floating action button (with a +) on the budgeting screen will be used to add a transaction to the current week’s ArrayList. By pressing that button, the user is brought to another activity whose title is “Add Transaction.” There are two blatantly obvious buttons: “MONEY IN” and “MONEY OUT.” Pressing MONEY IN brings the user to a different activity than pressing MONEY OUT. The Money In activity is a basic form where the user types in values for the Transaction’s Label, Amount (in $), indicate whether or not it is a paycheck, and add a tag (if wanted). The “ADD” button below all this will then create a new transaction with all of those fields and place it in the week’s list of transactions. The Money Out activity is very similar but will have some added functionality. The user still inputs a label, amount, and a tag, but by clicking the “Bill?” checkbox, another checkbox is brought into view: “Loan?” If the user ticks this box as well, a few more fields appear. One can enter the total balance of the loan and the APR interest rate, and then by a press of the “Calculate Loan” button, the table below it that was once empty (other than the top row with text) has its second and final row filled with numbers. The bottom-left cell of the table will have a number whose value is determined from a function call that calculates how many months it would take to finish the loan, given the principle, interest rate, and the Transaction’s Amount as the monthly payment. The bottom-left cell of the table will call a different function that determines the minimum monthly payment needed to finish off the given loan within 12 months. For the actual transaction, pressing the “ADD” button only then stores the value that the user inputted regarding the label, amount, tag, and whether or not it was a loan/bill. Pressing the ADD button in either of the Money In or Money Out activities returns the user to the budgeting activity.

From the budgeting activity, pressing the hamburger button in the top left of the screen will open a navigation drawer. Pressing the “Budgeting” option would simply close the drawer. Pressing “Investment Mapping” would bring the user to the Investment Mapping Activity (which will soon be discussed). Pressing “Settings” would bring the user to the Settings Activity. Only the root Budgeting, Investment Mapping, and Settings Activities would have the hamburger button and navigation drawer; every other activity

The investment mapping activity would have a few input fields with which the function called by the “Calculate” button would call a function that replaces the empty space in all the cells of the right-hand column of the table below it (aside from the top cell with text which both initially and finally has that same text) with numbers corresponding to how much any security with given principal amount, interest rate, and weekly contribution will grow over time.

The Settings Activity will show only two parts for now: “Delete Tags” and “About.” Pressing “Delete Tags” will bring the user to an activity where pressing an X next to any of the tags (tags will be added dynamically by checking each transaction of each week and adding it to the Activity’s ArrayList of tags if it is not already in it). Pressing the X also removes the tag from any Transaction that had that tag at the time. If the user instead presses “About,” a dialog is brought up that displays some information about the project, developer (that’s me!), and the app’s version.

This is kind of a lot of functionality to employ (it seems), but I believe that I’ll be able to do it well and on time because this project is something that I’d personally like to be able to use as soon and as effectively as I can.