# FINFREE: A System for Financial Planning, Monitoring, and Budgeting

e.g., PANTRACK: A System for Tracking Pandemics in Iceland

# Aidan Duffy

# Assignment 1 --1/12/2021 7:08 AM

For this assignment, you will describe and implement release 1 of your term project. You will incorporate *an abstract class*, *inheritance*, *upcasting or downcasting*, and *polymorphism*. You are free to choose a project that interests you but if you prefer, the instructor and your facilitator will be happy to suggest a topic. If you are already an experienced developer, this is an opportunity to build a challenging application (check with your facilitator if it requires significant API’s) or discuss research with the instructor. It is OK to name a project with much more scope than you can accomplish in the course (as in the example above): we will not hold you to completing everything associated with it. What we do expect is that you specify and implement a set of do-able requirements within such scope.

Submit this completed Word document. Replace as indicated. Please observe and retain the gray text. Your materials—in black 12-point Times New Roman—should not exceed 5 pages excluding references, figures, and appendices. Use the Appendix sections for additional material if you need to. These will be read only on an as-needed basis.

We want you to develop in Eclipse preferably or else IntelliJ (talk to your facilitator about exceptions). As you code, use JUnit tests whenever possible but certainly by week 2—package-by-package, class-by-class, and method-by-method, except for trivial methods and those requiring I/O. Use non-Junit classes for testing the latter. Keep the evaluation criteria in mind, listed at the end.

For this assignment, you do not need to read data from a file—you can build all data into the code.

Include a ReadMe file describing where to run the application from, and including necessary execution notes. All JUnit tests will be assumed runnable.

# 1.1 SUMMARY DESCRIPTION *EVALUATION CRITERION (i) APPLIES*

One- or two-paragraph overall description of your proposed term project—half-page (12-point Times New Roman) limit. By the end, term projects will incorporate most of the techniques discussed in the course. To do this, you may need to alter the direction of your project or introduce an additional project in future. You may alter this or even replace it as the semester progresses. You will probably find it useful to use your project acronym.

This project concerns an application for younger, less financially savvy adults that would enable them to develop their own financial freedom through effective budgeting and planning. The application, called FinFree, incorporates information and data from the user’s bank, brokerage, and credit card accounts in order to help give them a brief overview of their current financial position. The earliest iterations will ask the user to manually input this information, then later releases will ask for login information. Eventually, FinFree will incorporate Plaid as a login mechanism. From there, to use FinFree, users can preset budgeting goals, like saving for a big trip or just generally increase the size of their savings account or IRA. FinFree will also ask the user to set predetermined spending limits in a variety of core categories, such as necessities like groceries as well as discretionary purchases like coffee or alcohol, in an effort to help their budgeting. Alerts would be sent out if they overspend in any category.

## 1.2 I/O EXAMPLE FROM *PROJECTED* COMPLETED PROJECT *EVALUATION CRITERION (i) APPLIES*

Provide an example of projected *concrete* output for designated input. You will not be held to fulfilling exactly this—it is just explanatory at this point, to indicate where your project is going. We recognize that project direction and details will change as the term progress. This section refers to the project as a whole, not just to what you will produce this week, so we can gain an idea of what you have in mind overall.

FinFree would start at a menu with a number of options for the user to navigate through to see their current financial posture.

Here are two examples of a simple walkthrough in FinFree.

**FinFree:**

Welcome to FinFree, Aidan! Please select from one of the options below:

1. Check monthly net income
2. Check net income year to date
3. Check monthly spending
4. Check net worth
5. Check bank account balance
6. Check brokerage account value
7. Check credit card account balance

Ex 1:

**User:** 1

**System:**

You have been paid $1700 since the start of the month. You have spent $900 since the start of the month.

Your monthly net income is $800.

Ex 2:

**User:** 7

**System:**

You have 3 credit cards. Please select which card whose balance you are checking.

1. Chase Freedom
2. American Express Gold
3. Bank of America Cash Rewards

**User:** 2

**System:**

You have spent $452.81 on your American Express Gold card so far this month, and you have paid $0.00 off this balance. Your current balance on this card is $452.81.

## 1.3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE *EVALUATION CRITERION (ii) APPLIES*

Supply [functional requirements](https://docs.google.com/document/d/1eU7eINLDxmrf793D4OF2yGT4ry_SW3GQGoVDYzecGHc/edit?usp=sharing) statement that you accomplished for this assignment, i.e., functionality that the application provides for the user. Please state requirement in declarative form, as illustrated in the examples, because here we want to know the functionality intended (*what*, not *how*). For example, the following is *not* a proper functional requirement: *TicTac will have a class for O’s and a class for X’s.* It is common to mistake design elements like this for requirements. To get started, state what the application will accept as input, like requirement 1.3.1 below.

Keep in mind that the implementation of your requirements will incorporate *an abstract class*, *inheritance*, *upcasting or downcasting*, and *polymorphism*; that will probably influence the requirements you choose to implement in this assignment. The example material supplied should be deleted before you submit.

### 1.3.1 Create Accounts

FinFree shall accept a variety of accounts from the user (bank, credit card, brokerage, etc.) as an input.

### 1.3.2 Report Account Balances

FinFree shall report the value of the assets contained in all of the accounts.

## 1.4 ILLUSTRATIVE OUTPUT FROM IMPLEMENTATION *EVALUATION CRITERION (ii) APPLIES*

### Provide illustrative output from your implemented application (so far) showing that the requirements have been met. Explain what class.method(s) produce it.

Your response should replace this, as well as the example below.

The following is produced by *FragmentDisplayExample.displayFragmentHierarchyExample()*. *HeadingFragment* and *NonHeadingFragment* objects are hard-coded data.

=========FragmentDisplayExample==========

========= Should display 8 items ==========

My Expenses Heading

Taxes Heading

Estate taxes

Income taxes

Housing Heading

Mortgage for housing

Maintenance for housing

Food

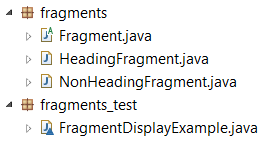
3 headings and 5 others.

## 1.5 YOUR DIRECTORY

### Show a screenshot of your directory. This should include a parallel directory of JUnit tests where possible—package-by-package, class-by-class, and method-by-method, except for trivial ones.

Your response should replace this, as well as the example below.

All methods in the K-Grade example are trivial so far except for ones that perform displays, and these are not suitable for JUnit testing.



## 1.6 TECHNIQUES IMPLEMENTED *EVALUATION CRITERION (iii) APPLIES*

Your implementation should include *inheritance*, *polymorphism*, and *either an abstract class or interface* at least once, and in a manner that is useful to your application. Explain where and how you applied these, using the headings below.

### 1.6.1 Class model and Sequence Diagram

Identify where you included *inheritance*, *polymorphism*, and *abstract classes* or *interfaces* in your class model. Make classes and members *static* or not as per their intended usage. To do this use tools (e.g., Visio and Lucidchart), PowerPoint, or a combine models as in [this example](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing) (which you are free to cut and paste from). Insert indications in red (as in [this example](https://docs.google.com/spreadsheets/d/1ZvkerE9FkWHWwVGdzuy7YMBU6oBMFGZbA4sotFETs8Y/edit?usp=sharing)) to show where the three features below apply.

The UML class model for the accounts is [here](https://lucid.app/lucidchart/invitations/accept/10326096-486b-4e00-b852-09bbf57adea2). *Account* is abstract. Polymorphism occurs in the versions of *credit*() and *debit*(). This is the case specifically for CreditCardAccounts, because one of these accounts can contain multiple cards, and the system must credit or debit the correct credit card. The figure includes inheritance. Here is an image of the UML:

Diagram

Description automatically generated

### 1.6.2 Code showing an abstract class or interface

Show the relevant code (only) and explain why an abstract class or interface is appropriate here. It should be clear where the code is located (class and method).

There are only three kinds of accounts, so there is no need for the *Account* base class to be concrete since there will be no generic Fragments. Additionally, these three types will have different implementations of these methods.

Text

Description automatically generated

### 1.6.3 Code showing polymorphism

Show the relevant code (only) and explain why *polymorphism* is appropriate here. Recall that polymorphism is implemented in one of two ways – overriding methods in subclasses or overloading methods in the same class where the method signatures are different – and allowing the language runtime to dynamically invoke the correct method. It should be clear where the code is located (class and method).

Your response should replace this, as well as the example below.

The method *displayTypeCount*() iterates through Fragment’s and calls *getType*() on each. This is polymorphism. The relevant code is shown below.

**static** **void** displayTypeCount() {

// Postcondition: The counts of heading- and non-heading type fragments in

// theFragments are on the console

**int** headingCount = 0, nonHeadingCount = 0;

**for** (Fragment fragment : *theFragments*) {

**if** (fragment.getType() == "Heading fragment") {

headingCount += 1;

}

**else** {

nonHeadingCount += 1;

}

}

…

}

### 1.6.4 Code showing upcasting or downcasting

Show the relevant code (only) and explain why upcasting or downcasting is appropriate here. It should be clear where the code is located (class and method).

Your response should replace this, as well as the example below

Some of the sub-fragments of a *HeadingFragment* are themselves *HeadingFragment*s. We need to cycle through all sub-fragments but identify *HeadingFragment*s so we can call *displayAll*(). Downcasting is needed from *Fragment* to *HeadingFragment*.

**public** **class** HeadingFragment **extends** Fragment {

**private** ArrayList<Fragment> theSubFragments = **new** ArrayList<Fragment>();

**….**

**public** **void** display() {

System.***out***.println(text);

}

**public** **void** displayAll() {

display(); // the heading itself

// display sub-headings and their sub-headings

**for**(Fragment fragment: theSubFragments) {

**if** (fragment **instanceof** HeadingFragment) {

((HeadingFragment)fragment).displayAll();

}

**else** {

fragment.display();

}

}

}

## 1.7 YOUR CODE

Unless your facilitator requests another method, copy your Eclipse project to your file system, zip it, and attach it. Please contact your facilitator in advance if you want to request an alternative means.

<Your response here>

## 1.8 EVALUATION OF ASSIGNMENT 1



## Appendix 1 (if needed; should be referenced above, and will be read as-needed only)

## Appendix 2 (if needed; should be referenced above, and will be read as-needed only)