# Homework 4

# CSS 161 : Fundamentals of Computing

# Instructor Rob Nash

## Summary

Continue to develop your understanding of ***Repetition Control Structures*** and ***File IO*** by building a class averaging program. This software will read input data from a file describing specific classes and students, and will calculate weighted averages for each student (as well as a pass/fail mark).

## Work Items

1. Submit only your java program (“.java” text file) via the website’s dropbox.
2. Pay extra attention to code formatting, style, comments, and basic programming practices learned so far.

## The Class Averaging Program

Your assignment is to write a program that outputs a summary of classes and students, given input data in a specific format (more on that in a bit). For a high-level view, look at the sample program execution below. Your program will read from an input file (“courseData.txt”) a number of data items, stored in the file in the following order:

1. Global data is on the first line, as this applies to all classes. This first line holds the weights to use for ***all*** classes, in this order: program weight, midterm weight, and final exam weight.
2. Every other line in the file is specific to a class, and so the next line starts with a course identifier (161, 162, or 263, and this is an int).
3. After the course identifier, a set of lines belonging to that course are listed, each with a student ID (four digits, such as 1234), an unweighted program score, an unweighted midterm score, and an unweighted final score.
4. Reaching a 0 indicates the end of input for a specific class, or the end of file if all class data has been consumed.

## Sample Input

For example, the following data comes from the sample file you’ll use as input to test your software:

0.30 0.30 0.40

161

3333 70 60 50

4444 50 50 50

5555 80 90 80

0

162

1212 90 85 92

6666 60 80 90

…

…

…

0

## Sample Output

This section outlines what your software is to calculate and ultimately report on the console (or, for added challenge, in a GUI using JOptionPane).

Grade Data For Class 161

ID Programs Midterm Final Weighted Average Programs grade

-- -------- ------- ----- ---------------- --------------

3333 70 60 50 59.00 Pass

4444 50 50 50 50.00 Fail

5555 80 90 80 83.00 Pass

Class Average: 64.00

Grade Data For Class 162

ID Programs Midterm Final Weighted Average Programs grade

-- -------- ------- ----- ---------------- --------------

1212 90 85 92 ... Pass

6666 60 80 90 ... Fail

7777 90 90 90 ... Pass

8888 95 87 93 ... Pass

9090 75 77 73 ... Pass

Class Average: ...

Grade Data For Class 263

ID Programs Midterm Final Weighted Average Programs grade

-- -------- ------- ----- ---------------- --------------

2222 . . .

8989 . . .

9999 . . .

Class Average: ...

## Notes

* Use the skeleton code below as a starting point for your project, as it outlines a well-structured and nicely commented program.
* The columns “Programs”, “Midterm”, and “Final” above are raw scores (not weighted). The only calculation that involves manipulation of weights and grades is the weighted average. The ***Weighted Average*** is calculated using the following sum of products:

(programsWeight\*programsGrade) +(midtermWeight\*midtermGrade) +( finalWeight\*finalGrade)

* The ***Pass/Fail*** determination is made based only on the raw score of the students programs. If the student’s programs score is greater than or equal to 70, they pass. If less than 70, they fail.
* Also, your program will have multiple loops (one per course, one per student, etc.) *which you should ignore when initially tackling this problem.* When solving this problem, it may be best to focus first on the innermost loop that deals with students. Deferring higher level details in favor of fleshing out individual cases first can be thought of as an inside-out approach (commonly called ***Bottom-Up Design***, related to ***Stepwise Refinement***). This is similar to ignoring a forest and paying attention to just one tree. For example, you could start by focusing on just one class first and building the loop to process each student until a 0 is reached. Once this is working for one class, you can wrap this logic inside of a loop and extend its functionality to process multiple classes.
* Finally, the real data may differ from the sample data with respect to actual scores used, but will not vary with respect to file format. This simply means that your program should work with multiple files formatted in the same fashion, and not just with the values in the sample file.

## Program Requirements

There exists many ways to build this code, but the requirements stay the same. Your software should:

1. Comments! I’ve included a number of comments in the sample code below as examples. This is the first assignment where you’ll be graded on both correctness and code quality (wrt comments).
2. Read in weights, IDs, course numbers, programs scores, midterm scores, and final scores per student from the sample input file provided (using Scanner or FileReader).
3. Calculate statistics per student and per class and report the following:
   1. Output a weighted average per student
   2. Output a Pass/Fail mark per student
   3. Output an average per class
4. Your output should be compared against the sample output above for accuracy.

## Hints

* Consider the code below for a great starting point.
* Don’t wait till the last minute to get started or get help .
* Use comments in your code, since this time it will be graded!
* Break up this program into smaller functions and call those
  + You can call a function inside of a loop

**import** java.util.Scanner;

**import** java.io.File;

**import** java.io.IOException;

//authors: Fukuda, Zander, (edited by Nash)

**public** **class** ClassAverage {

//... class constants go here

**public** **static** **void** main(String[] args) **throws** IOException {

**int** courseNumber; // number of the course

Scanner inputFile = **null**; // file containing data

inputFile = **new** Scanner(**new** File("data.txt"));

// ... any stuff you need to do one time

//Per class, print a table of ID numbers, grades, weighted average

// per student, and a Pass or Fail programs grade.

// The class average is also printed.

**for** (...) {

// Read class number, print class number title, headings.

courseNumber = inputFile.nextInt();

... rest of the code goes here

// initialization

... code goes here

// Loop to handle one class.

//For each student in the class, get and print their information,

// compute their avg, and sum the avgs

**while** (...) {

... code goes here

}

// compute and print class average

//... code goes here

}

}