

## CS 235 - Homework 3

### Deadline:

Due by **11:59 pm on Wednesday, September 16, 2015.**

### How to submit:

Submit your files using `~st10/235submit` on nrs-projects, with a homework number of 3, by the deadline shown above.

### ***Reminder: Instructions for using the tool `~st10/235submit`:***

- If you did not create your files (using, for example, `nano`) on nrs-projects, then first transfer your files to be submitted to a directory on nrs-projects.
  - On Linux or Mac OS X, you can use `sftp` (secure file transfer) to connect to `nrs-projects.humboldt.edu`, and then transfer them
  - In a Windows lab on campus, you can use the **WinSCP** program, which is a graphical `sftp` program.
  - The free program **FileZilla** is a graphical `sftp` program that works on Windows and Mac OS X (and maybe also Linux?)

- Once your files to be submitted are in a directory on nrs-projects, then use `ssh` (or PuTTY in a Windows lab on campus) to connect to `nrs-projects.humboldt.edu`.

- Use `cd` to change to the directory containing the files to be submitted -- for example,

```
cd 235hw03
```

- Type the command:

```
~st10/235submit
```

...and give the number of the homework being submitted (or whatever number you have been asked to do for lab-related files) when asked, and answer that, `y`, you do want to submit all of the files with of-interest-to-235 suffixes in the current directory. (Note that I don't mind if a few extraneous files get submitted as well -- I'd rather receive too many files than too few, and typing in all of the file names for some assignment is just too error-prone...)

- You are expected to carefully check the list of files that the tool believes have been submitted, and make sure all of the files you hoped to submit were indeed submitted! (The most common error is to try to run `~st10/235submit` while in a different directory than where your files are...)

### Purpose

To practice writing another non-application class along with a subclass, and exercising them a bit.

## Important notes:

- Follow the initial class Java coding standards mentioned in class -- some of these include:
  - Follow the Java naming standards that have been discussed in class.
  - Attempt "javadoc-style" comments for **each** Java class and method, in the same style as you see in posted in-class examples.
  - Everything inside a set of { } must be indented by AT LEAST 3 SPACES -- and the beginnings of statements that are sequential should be indented the SAME NUMBER of spaces. (That is, sequential statements should line up.)
  - { and } should each go on their OWN line, with { lined up evenly with the preceding line, with the { }'s contents indented by at least 3 spaces, and with } lined up with the opening {. That is, handle the curly braces as you see in all posted class examples!
- ASK ME if any of these are unclear to you!

## Problem 1

Consider a cooking school, named Cookery. It has a mailing list. Any person can be on the mailing list. Design and implement a public class `CookeryFan` that meets the following criteria:

- it has private data fields for the person's first name, last name, preferred e-mail address, a four-digit year they were added to the mailing list, and whether they are willing to receive fund-raising e-mails in addition to information e-mails about school events
- it has at least one constructor, a 5-parameter constructor that allows initial values of all of these data fields to be initialized
- it has an appropriate accessor/getter method for all five data fields
- it has an appropriate modifier/setter method for all of the data fields except the year one was added to the mailing list -- that cannot be changed by users of this class
- it overrides the `toString` method to give an appropriate `String` depiction of a `CookeryFan` instance
- add at least one additional method of your choice to this class as well
  - (For example, perhaps a method that produces a `String` summarizing certain of a fan's information,
  - or a method that returns approximately how many years a fan has been on the mailing list)
  - These may be shared with the class on the course Moodle site.

Submit your resulting `CookeryFan.java`.

## Problem 2

Write a Java application in `CookeryFanPlay.java` that exercises your `CookeryFan` class from Problem 1. It should meet at least the following criteria:

- It should create at least two `CookeryFan` instances of your choice.
- It should print to standard output a message saying that it is going to call the `toString` method for both `CookeryFan` instances, and then print to standard output the result of doing so.
- It should print to standard output a blank line and then a message saying that it is going to call all five of `CookeryFan`'s accessor/getter methods for one of the `CookeryFan` instances, and then print to standard output the results of doing so.
- It should print to standard output a blank line and then a message saying that it is going to call all four of `CookeryFan`'s modifier/setter methods for one of the `CookeryFan` instances, and then it should do so,  
and then print the result of calling the `toString` method for that `CookeryFan` instance (to show how its state has changed as a result).
- It should print to standard output a blank line and then a message saying that it is going to call your additional method(s) for at least one of your `CookeryFan` instances, and then print to the screen something indicating the result of that call/those calls to your additional method(s).

Submit your resulting `CookeryFanPlay.java`.

## Problem 3

It turns out that some of the people on the Cookery cooking school mailing list are also students at the school. Design and implement a class `CookeryStudent` that is a subclass of `CookeryFan`, with the following additional characteristics:

- a `CookeryStudent` instance should also have private data fields for a GPA (based on the usual 4-point scale), and a culinary specialty. Include an appropriate accessor/getter method and modifier/setter method for each of these two additional data fields.
- include a 7-parameter constructor expecting the appropriate initial values for the desired new `CookeryStudent` object's first name, last name, preferred e-mail address, 4-digit year added to the mailing list, whether they are willing to receive fund-raising e-mails, 4-point-scale-based GPA, and culinary specialty.
- include a method `onHonorRoll`, which determines whether the student's GPA is greater than or equal to 3.5. (Use a named constant appropriately for this "honor roll boundary".)
- override `toString` appropriately for this class, including GPA and culinary specialty in the resulting `String` (along with first name, last name, preferred e-mail address, 4-digit year added to the mailing list, and whether they are willing to receive fund-raising e-mails).

Submit your resulting `CookeryStudent.java`.

## Problem 4

Write a Java application in `CookeryStudentPlay.java` that exercises your `CookeryStudent` class from Problem 3. It should meet at least the following criteria:

- It should create at least three `CookeryStudent` instances of your choice, except one should have

a GPA less than 3.5, one a GPA equal to 3.5, and the other a GPA greater than 3.5.

- It should print to standard output a message saying that it is going to call the `toString` method for all three `CookeryStudent` instances, and then print to standard output the result of doing so.
- It should print to standard output a blank line and then a message saying that it is going to call all of the accessor/getter methods for one of the `CookeryStudent` instances, and then print to standard output the result of calling all **seven** of them (the five inherited from `CookeryFan` and the two new ones added to `CookeryStudent`).
- It should print to standard output a blank line and then a message saying that it is going to call `onHonorRoll` for each of the three `CookeryStudent` instances, and then decide on an appropriate way to output to the screen messages clearly giving the results of those calls.
- It should print to standard output a blank line and then a message saying that it is going to call all six of the modifier/setter methods for one of the `CookeryStudent` instances, and then it should do so (calling the four inherited from `CookeryFan` and the two new ones added to `CookeryStudent`), and then print the result of calling the `toString` method for that `CookeryStudent` instance (to show how its state has changed as a result).

Submit your resulting `CookeryStudentPlay.java`.