600.107 Introductory Programming in Java, Summer 2015
Homework 8
Due: Friday, July 24th at 11:59pm

Please review the **Homework Guidelines** handout for this course before you begin.

> **Restriction:** For this assignment, you may certainly use arrays, but are NOT permitted to use ArrayLists.
>
> Repetition Statements: "while", "for", or "do-while"- type statements MAY be used in this exercise, but you are NOT permitted to use `break;` or `continue;` statements except that you may use `break;` inside a case within a `switch` statement.

Task. In this assignment, you'll be creating several files to model a restaurant menu. We have provided Javadoc documentation for two classes that you must write: `MenuItem` and `Menu`. You'll need to implement each described data member, constructor, and method exactly as specified in that documentation. Additionally, you must embed Javadoc comments into your source files that actually can be used to re-generate the provided documentation. Please read the descriptions of each member (i.e. data field, constructor, or method) carefully, as there are several implementation-related requirements for the assignment listed there. But, to orient you, here is a very brief summary.

The `Menu` class defines a menu object, and, in general, a `Menu` will eventually contain a number of individual `MenuItem` objects. So, you'll want to write the `MenuItem` class first, and test its methods carefully. Next, write the `Menu` class, and test its methods. To help in the testing process, we're providing for you a `Restaurant` class which has a driver program that implements a menu of choices (no pun intended) for the user to interact with a `Menu` object. You should not modify the `Restaurant.java` file, but the classes you write must work with it. You will not submit `Restaurant.java` as part of the assignment. In fact, you don't need to submit any file that contains a `main` method! We will run your file with our own driver program's `main` method when grading it.

The two files that you write do not need to collect any keyboard input. In the `Restaurant` driver program, all input is given by the user at the command-line. To help you get a sense of the assignment and what an execution of the `Restaurant` driver program will look like, a text file containing a sample run of the `Restaurant` program is posted on Piazza. Aim for the output given by the `Restaurant` driver program when it is run with your `MenuItem` and `Menu` classes to match the sample output exactly when given the same input. Be sure to test on other inputs, too!

Hints. You've already had a bit of practice writing instantiable classes, so I won't repeat the HW7 "Tips" section here, though you might find much of it relevant for HW8.

The new skill needed for this assignment is working with an array. **Sketching a picture of the array and its contents at any given time may be really helpful as you work on this assignment.** As the list of menu items is built up slowly by the user's repeated calls to `addItem`, the array `menuItems` in the `Menu` class is not necessarily full at any particular time. The strategy for working with *partially-filled arrays* like this one almost always involves holding all actual data values in contiguous slots in the array, starting at position 0. Then, the unused slots of the array, when there are any, are always gathered near the bottom of the array.

Therefore, the `int` instance variable `numMenuItems` is maintained by the class to allow it to keep track of the location of the next available position in the array. That is, the next item added to the menu should be placed at the position of the array called `numMenuItems`. This is `int` index `numMenuItems` is also crucial for any traversal of all menu items in the array, such as when all items need to be output or checked for calorie counts, etc. Traversals like that through the list of menu items

should never "fall down" past the end of the portion of the array that actually contains menu items, since later positions will contain null pointers rather than actual menu items (and calling methods on null pointers causes `NullPointerException` crashes!). That means that rather than going through all items from position 0 up to (but not including) the array's actual length, traversals should generally go from position 0 up to (but not including) the value of `numMenuItems`.

Additional care needs to be taken when an item is removed from an array. If the item to be removed is in the middle of the array, we shouldn't just remove the item and leave a "hole" there. Instead, items from later positions in the array are generally moved up to fill the vacant spot. In the case of this menu array, there are no requirements to keep menu items in a particular order, so the most efficient way to fill the vacancy is to simply move the last menuItem in the array up to fill the vacancy. (That is, move the item that resided at `numMenuItems - 1` prior to the method call directly up into the vacated spot.) This avoids the need to shift a number of other elements up, one position each, which is what we would have needed to do if we were required to maintain the ordering of the remaining items.

The value stored in `numMenuItems` is also useful in determining when the array is so full that its size needs to be increased. Once `numMenuItems` matches the length of the array, no more items will fit in the list! This is your clue to *resize* the array, which will include instantiating a larger array, and copying all existing menu items over into the new, larger array. To complete the resizing process, don't forget to set the `menuItems` reference to hold the address of the larger array. (By the way, although the array's size must be increased when it gets very full, this program doesn't decrease the size of the array when it gets closer to empty.)

And finally, here's one last suggestion. You must implement all methods listed on the Javadocs, but you may add additional methods. You are free to add helper methods to your classes if you find yourself needing to perform the same task repeatedly. For example, when working in the `Menu` class, there are several places where you need to determine if a particular name already appears in the list of items. It might be nice to have a `private` helper method that takes a name and returns the position of that name in the array, or the special value `-1` to indicate that it is not present. This method could simplify the code you need to write in several other places in the class.

Turn-in. Before the due date listed at the top of this file, please submit via the course Blackboard site one complete `.zip` file named `HW8-jhed.zip` (where you replace 'jhed' with your own personal JHED) containing all `.java` files required for the above tasks. Be sure that all submitted code compiles! If you were unable to complete a task, please include as part of your `.zip` submission a text file named `README` explaining anything you'd like the graders to know.