CS 64 **Midterm Exam 1**

This exam is due at 5:00 p.m. on Thursday, 6 March 2014. By the deadline, you should hand in hard copy of your `.java` files **AND** a hard copy of the output produced in the Console View of Eclipse by running your code. **Also** by the deadline, you should drop an Eclipse project folder that is entitled with `your name` and "`Exam1`" and that contains your package of Java code into the `Drop Box Exam 1` within `/Users/cs64/Public` on `ellafitz.mills.edu`. This exam is open book, open notes, open assignments, and open labs. You will need the `Name` class that you have developed through homework assignments. If you have problems using your own `Name`class, I can provide my `Name`class for you to use to complete this exam.

You may **not communicate** with anyone other than the instructor, Barbara Li Santi, about this midterm exam until you receive an email from the instructor indicating that all of the exams have been submitted. **If you have been working with partners on the homework assignments, you need to be particularly careful about not communicating because you have become so accustomed to communicating about assignments.**

The programming goal of this midterm is to write Java code that implements the class `NameCollection` described below **and** to write a class named `TestNameCollection` where all of the constructors and methods of class `NameCollection` are tested.

Specifications for the class `NameCollection`

Objects of class `NameCollection` will have one `String` instance variable to represent the identifier for the `NameCollection`, one array of `Names` instance variable to hold references to the `Names` in the `NameCollection`, and one `int` instance variable to represent the `count` of `Names` in the `NameCollection`. The class `NameCollection` will have one `public static final int` variable representing the default maximum size of the `NameCollection` when none is provided as a parameter at the time of construction. The class `NameCollection` will have a **2**-parameter constructor, a **1**-parameter constructor, and **16** public methods:

a) The **2**-parameter constructor has `1 String` parameter representing the identifier for the `NameCollection` and an `int` parameter representing the maximum size of the `NameCollection`. The constructor assigns the `String` parameter to the instance variable that represents the identifier for the `NameCollection`, uses the `int` parameter to construct an array of `Names` that is then assigned to the array instance variable, and sets the `count` instance variable to `0`.

b) The **1**-parameter constructor has `1 String` parameter representing the identifier for the `NameCollection`. The constructor assigns the `String` parameter to the instance variable that represents the identifier for the `NameCollection`. This constructor uses the `public static final int` variable to construct an array of `Names` that is then assigned to the array instance variable, and sets the `count` instance variable to `0`.

c) An access method named `getIdentifier` for the identifier instance variable.

d) A modifier method named `setIdentifier` for the identifier instance variable that has one `String` parameter. If the `String` parameter is an empty `String`, an error message should be displayed using `System.out.println`, and no assignment should be made to the `String` instance variable. In the case that the `String` parameter is not an empty `String`, it should be assigned to the `String` instance variable.

e) An access method named `getArray` for the array of `Names` instance variable. The return type for `getArray` should `Name[]`.

f) A modifier method named `setArrayAndCount` for the array of `Names` instance variable and the `count` instance variable. This modifier method has two parameters. The first parameter is a reference to an array of `Names`. The second parameter is an `int` representing the count of `Names` in the array parameter. The method should test to determine whether the parameter representing the count of `Names` in the parameter array is greater than the size of the parameter array. If this is the case, an error message should be displayed using `System.out.println`, and no assignments should be made to the instance variables. In the case that the count parameter is appropriate for the size of the array parameter, the `count` and array instance variables should each be assigned the values of the corresponding parameters.

g) An access method named `getCount` for the `count` instance variable.

h) A method named `getElementAtPosition` that has return type `Name` and one `int` parameter representing an index in the array. If the parameter is not a valid index for the array, an error message should be displayed using `System.out.println`, and the value `null` should be returned. If the parameter is a valid index but refers to a location in the array beyond the last `Name` reference at index `count - 1`, an error message should be displayed using `System.out.println`, and the value `new Name ("", "", "")` should be returned. In the case that the parameter is a valid index of a `Name`, the reference to that `Name` in the array should be returned.

i) A `boolean` valued method named `changeElementAtPosition` that has one `int` parameter and one `Name` parameter: the first parameter is an index in the array and the second parameter is a reference to a `Name` object to be stored in that index of the array. If the first parameter is not a valid index for the array, an error message should be displayed using `System.out.println` and the value `false` should be returned. If the parameter is a valid index but refers to a location in the array beyond the last `Name` reference at index `count - 1`, an error message should be displayed using `System.out.println`, and the value `false` should be returned. In the case that the parameter is the index of a `Name` reference in the array, the second parameter should be assigned to the index in the array indicated by the first parameter, and the value `true` should be returned.

j) A `boolean` valued method named `equals` that overrides the `equals` method of the `Object` class. The `equals` method takes an `Object` parameter that it casts to a `NameCollection` object. The `equals` method returns `true` if and only if the identifier for the invoking `NameCollection` object is identical to the identifier for the parameter `NameCollection` object, the `count` of `Names` in the invoking `NameCollection` object is equal to the `count` of `Names` in the parameter `NameCollection` object, and the invoking `NameCollection` object and parameter `NameCollection` object refer to the same array of `Names`.

k) A `toString` method that returns a `String` representation of the information in an object of class `NameCollection` in the following format: the identifier for the collection followed by a space, followed by the `count`, followed by a colon and a new line character, followed by the `Names` stored in the collection separated by tab characters and with a new line character after each group of 3 `Names`.

l) A `boolean` valued method named `addDataItem` that takes a `Name` parameter that will be an additional `Name` in the `NameCollection`. If the `count` of `Names` in the collection is less than the size of the array instance variable, the `Name` parameter should be assigned to the next available index in the array, the `count` instance variable should be incremented, and the value `true` should be returned by the method. If the `count` of `Names` in the collection is equal to the size of the array, the value `false` should be returned by the method.

m) A `boolean` valued method named `isFull` that returns `true` if the collection is full and cannot hold another `Name`.

n) A `boolean` valued method named `isEmpty` that returns `true` if the `count` of `Names` in the collection is `0`.

o) A method named `sort` that sorts the `Names` stored in the collection into alphabetical order.

p) An `int` valued method named `indexFirstOccurrence` that takes a `Name` parameter and returns the index in the array of the first occurrence of the parameter or returns `-1` if there is no `Name` in the array equal to the `Name` parameter.

q) A `boolean` valued method named `deleteFirstOccurrence` that takes a `Name` parameter and attempts to delete the first occurrence of the `Name` parameter from the collection. In the case that the deletion occurs, all elements in higher numbered indices in the array should each be moved to the next smaller numbered index, the `count` instance variable should be decremented, and the value `true` should be returned by the method. If no occurrence of the `Name` parameter is found, the method should return `false`.

r) A `boolean` valued method named `deleteAllOccurrences` that takes a `Name` parameter and attempts to delete all the occurrences of the `Name` parameter from the collection. In the case that any deletion occurs, for **each** deletion of a `Name`, all array elements in higher numbered indices should each be moved to the next smaller numbered index, and the `count` instance variable should be decremented. Finally, the value `true` should be returned by the method. If no occurrence of the `Name` parameter is found, the method should return `false`.