[**Department of Computer Science**](http://www.cs.umd.edu/)

[**CMSC131:**](http://www.cs.umd.edu/class/fall2017/cmsc131-FC01/) Fall 2017

**Project:** Name Class

**Due Date:** Tue Oct 10, 8 pm / Fri Oct 13, 8 pm

## Overview

For this project you will implement a class called **Name**. This will allow you to practice constructors, get/set methods, instance variables, static variables and static and non-static methods.

**There are two deadlines associated with the project. Those deadlines are:**

**Tue Oct 10, 8:00 PM**

Your code must pass the public tests. That is the only requirement for this deadline. We will not grade the code for style. This first part is worth .5% of your course grade (NOT .5% of this project grade). Notice you can still submit late for this part.

**Fri Oct 13, 8:00 PM**

Final deadline for the project. Notice you can still submit late (as usual).

## Grading

* (10%) Public Tests
* (72%) Release Tests
* (8%) Secret Tests
* (10%) Style

## Code Distribution

The project's code distribution is available by [checking out the project](http://www.cs.umd.edu/eclipse/manage.html) named **NameClassProj**. To see the files click on the Eclipse triangle that is next to the **src** icon of your project. The code distribution provides you with the following:

* A file named **Name.java** - This is the class you need to implement.
* A file named **PublicTests.java** - This represents the public tests for the project; do not modify this file.
* A file named **Driver.java** - Illustrates how to use some of the methods of the class.

Notice that the Name.java class can be found in the **sysutilities** package. Remember that a package is similar to a folder. Just click on the Eclipse triangle you will see next to the sysutilities package and you will see the Name.java file. The PublicTests.java can be found in the tests package.

## Specifications

Your task is two implement the **Name** class. This class represents a person's name. It has three instance variables representing the first, last and middle name (all are string variables). A character instance variable representing a separator (to be used for printing purposes) is also part of the class. The only valid separators we can have are a comma (,), a dash (-), and a pound symbol (#). In addition a nickname instance variable (string) keeps track of the person's nickname (if any).

For this class you need to define a private method called validSeparator that takes a character as a parameter. The method will return true if the character is one of the three valid separators and false otherwise. Make sure you use this method when you need to validate a separator. In addition, the class will keep track of the number of instances created by using a private static field called nameObjsCount. This field is initialized to zero when you declared it.

The methods associated with the class are provided below. All the methods are **non-static** unless specified otherwise. You may want to take a look at the files Driver.java, PublicTests.java and pubTest1.txt as they can help you clarify the signature associated with each method and the expected output format.

1. **Default Constructor -**  Initializes a Name object with "NOFIRSTNAME", "NOMIDDLENAME", "NOLASTNAME" as first name, middle name and last name, respectively. In addition, the separator will be the '#' character.
2. **Constructor with two parameters -**  The first parameter is the first name and the second the last name. The middle name will be the empty string and the separator will a comma.
3. **Constructors with three parameters -**  The parameters are the first name, the middle name, and the last name. The separator will be a comma.
4. **Constructors with four parameters -**  The parameters are the first name, the middle name, the last name and the separator. If an invalid separator is provided, a comma will be used instead.
5. **getFirstname -**  Get method for first name.
6. **getMiddlename -**  Get method for middle name.
7. **getLastname -**  Get method for last name.
8. **setNickname -**  Set method for nickname.
9. **getNickname -**  Get method for the nickname. If there is no nickname the method returns the empty string. Notice the empty string is not the same as null.
10. **setSeparator -**  Set method for the separator. If the separator parameter is invalid, the separator for the Name object will not be changed.
11. **getSeparator -**  Get method for the separator.
12. **equals -**  Two names are considered equal if they have the same first, middle, and last names. The nickname is ignored.
13. **toString -**  Returns a string with the last name, the first name, the middle name (if any), and the nickname in parentheses (if any). Notice the separator must be used to separate each value. If there is no middle name, no separator should be added after the first name. The driver output below and the public tests provide examples of the output your program must generate.
14. **compareTo -**  It will return a negative value if the current object precedes the parameter in alphabetical order, 0 if the current object is equal to the parameter and a positive value otherwise. You must implement this method by comparing last names, then first names and middle names. **You MAY NOT implement this method by converting the Name object to string (using toString()) and then using the String class compareTo method on the resulting strings.**
15. **getNumberOfNameObjects -**  **Static** method that returns the number of objects that have been created.
16. **normalize -**  **Static** method that takes a Name object and a boolean as parameters. The method returns a **new** Name object where the first name, the last name and the middle name (if any) have been capitalized if the boolean parameter is true. If the boolean parameter is false all the names will be in lowercase. Use the string methods toUpperCase() and toLowerCase() for the normalization. Comma will be the separator for the normalized name object.
17. **validSeparator -** **Private** method that takes a character as a parameter and returns true if the parameter is a valid character. **You must use this method whenever you need to verify the validity of a separator. If you define but do not use the method you will lose credit (code duplication penalty).**

## Other

* For your project, the initialization of instance variables should take place in the constructors and not in at declaration time.

## Sample Driver

The following driver relies on the class you need to implement. You will find this driver in the code distribution.

**Driver**  
  
public class Driver {  
  
 public static void main(String[] args) {  
 Name name1 = new Name("Claudia", "I.", "Smith");  
 System.out.println(name1);  
  
 Name name2 = new Name("Rachel", "I.", "Green", '#');  
 System.out.println(name2);  
  
 Name name3 = new Name("Joseph", "K", "Falk");  
 name3.setNickname("Joe");  
 System.out.println(name3);  
  
 Name name4 = new Name();  
 System.out.println(name4);  
  
 System.out.println("Same: " + name1.equals(name2));  
  
 System.out.println("Comparing: " + (name1.compareTo(name2) < 0));  
  
 Name name5 = Name.normalize(name3, true);  
 System.out.println("Normalized: " + name5);  
  
 System.out.println("Total number of name objects: " + Name.getNumberOfNameObjects());  
 }  
}  
  
**Output**  
  
Smith,Claudia,I.  
Green#Rachel#I.  
Falk,Joseph,K(Joe)  
NOLASTNAME#NOFIRSTNAME#NOMIDDLENAME  
Same: false  
Comparing: false  
Normalized: FALK,JOSEPH,K  
Total number of name objects: 5

## Submission

Submit your project as usual. Make sure you check the results in the submit server. Remember that you need to do release testing to see the results of release tests.

## Academic Integrity

Please make sure you read the academic integrity section of the syllabus so you understand what is permissible in our programming projects. We want to remind you that we check your project against other students' projects and any case of academic dishonesty will be referred to the [Office of Student Conduct](http://www.jpo.umd.edu/).