# COP 3330
# Homework 3

## Out: 2/10/12 (Friday)
## Due: 2/24/12 (Friday) at 11:55 PM Webcourses time
**Late submissions accepted until 2/26/12 (Sunday) at 11:55 PM (with penalty)**

## Objective

1. To familiarize you with writing your own classes.
2. To help you become more comfortable with interactions among classes.
3. To familiarize you with the Comparable interface.

## Notes

The sample input and output provided for each problem is not comprehensive. When we grade your programs, we will test them using *different* data. You should test your program using different data, too.

## Code

You may discuss high-level ideas of how to solve these problems with your classmates, but do not discuss actual source code (except for boiler-plate code such as opening a file for reading, reading user input, etc.). All source code should be yours and yours alone. **Do not:**

- Share your source code with anyone
- Ask anyone to share his/her source code with you
- Ask/pay a programming forum/community to write your homework for you
- Pair program
- Do anything else that you know is deceptive, dishonest, or misleading in any way

## Problem : Assign That Program!

You've been hired by the CS department to help them make their system of distributing homework assignments to TAs to grade more efficient. Right now, each professor arbitrarily assigns homework assignments to his/her TAs, but the process is very inefficient. They would like you to write the backend for a system which takes in a list of submitted homework assignments, then allows TAs to make queries for homework assignments to grade. For now, you decide to just get a proof of concept working offline; you will worry later about how to take it online.

Your program should be split into two main components:

1) Processing the list of submitted homework assignments
2) Answering TA queries

### Processing List of Homework Assignments

You will be given an input file named `"HW_List.in"` containing a list of all submitted homework assignments for a given class. The list is composed of homework assignments for *all sections* of a given course. Due to the extreme buginess of Webcourses, the list is generated in a random order, and *may contain duplicate entries*. The input file will contain multiple homework assignments, one assignment per line. Each homework in the input file will be described by a line in the following format:

```
<Student Name> <Section number> <Date submitted> <Number of files> <List of file names>
```

The student name will always be in the form "`<Last Name> <First Name>`" (quotes for clarity). The section number will be an integer between 1 and 100. The date will always be an integral value between 1 and 31 inclusive (for simplicity, you may assume that all homework assignments are due on the 15th of every month, and homework assignments will always be submitted the month they are due). The number of files will always be an integer between 0 and 100, inclusive, and the list of file names will always contain exactly that many files. Each file name will consist of letters, digits, or symbols (but no whitespace characters), and each file name will be separated by a single space. Each part of the input will be separated by a single space, and is guaranteed to be well formed. You should continue processing homework assignments in the file until you reach the end of file.

For each homework that you process, you should print a line describing the homework to a file called `HW_Allocation.out`. Specifically, you should note whether each homework was or was not added to the queue for a given section. See the sample provided for the exact format of this output. You must **exactly** follow the format provided to receive full credit.

You should write a *Homework* class to encapsulate the information about a single homework assignment. It should consist of the name of the student who submitted the homework assignment, what section number this student is enrolled in, the *Files* which are submitted as part of this homework, the number of days that this homework is late (a negative value indicates the assignment is turned in early, 0 indicates the homework was submitted on the due date, and a positive value indicates it was turned in late), and a unique ID identifying this homework assignment which is different than the ID of any other homework assignment submitted. When allocating IDs to homework assignments, you should start at 1 and give each next homework assignment the next available unique id. You will inevitably assign new IDs to assignments which are duplicates and do not end up getting added to the queue due to already being there. This is fine.

In order to make sure grading is done in the proper order, your *Homework* class must be ordered in a very specific way. Namely, given two *Homework* assignments *A* and *B*, *A* should come before *B* in sorted order if:

1) *A* was submitted before *B* (earlier submitted homework get graded first), or
2) *A* and *B* were submitted on the same date, but the *Files* in *A* get sorted before the *Files* in *B*, according to the ordering of *Files* (see the *Files* class for the implementation of this ordering), or
3) *A* and *B* were submitted on the same date and have the same *Files*, but the student who submitted *A*'s name comes lexicographically before the student who submitted *B*. Name comparison is done first by comparing last name, and only if last names are the same then comparing first name. All string comparison is case sensitive.
4) If none of these hold true, then *A* and *B* are deemed to be the same *Homework* assignment.

The provided documentation shows all of the public variables and methods that must be implemented as part of your *Homework* class. You must implement all of these methods and variables, *even if you don't use all of them in your specific implementation*. **Failing to implement all details provided in the documentation will result in a decrease in your code score, even if your execution matches the judge output.**

You will be provided with a *HomeworkQueue* class which you should use to store all homework assignments that that you read from the input file. Reference the provided Javadocs for how to use this class. You should *not* make any modifications to this class.

You will also be provided with the *Files* class to use as part of the *Homework* class. You must not make any changes to the *Files* class.

You should implement the *Homework* class and the *Name* class, per the details of the specified documentation, and turn these in as part of your homework assignment.

**<u>Answering TA Queries</u>**

Once you have finished processing all homework assignments, you should handle answering TA queries. You will be given an input file named `"TA_Queries.in"` containing all TA queries which you should process. Each TA query will be of the form:

```
<TA id> <TA section> <Number of homework assignments requested>
```

The TA id is a unique number identifying this TA from all other TAs. TA section is the section number for which this TA should be grading assignments, and will be an integer between 1 and 100. The number of homework assignments requested is how many different homework assignments this TA is requesting to grade at this time, and will be between 1 and 1000.

All output should be printed to the file `HW_Allocation.out` (i.e. continue printing after the output of processing each homework assignment). Each query should output the next *k* homework assignments which are allocated to *this section* (in the sorted order described above), where *k* is the number of homework assignments requested by this TA. In case there are not *k* more homework assignments available to be allocated to this section, then you should allocate as many homework assignments as are left in the queue for this section to this TA, then print a message saying you could not allocate as many assignments as the TA requested. If there are no homework assignments for that section (either because there were no assignments turned in for that section or they have all already been removed from the queue), then you should print a message saying so. Follow the exact format of the sample output provided when assigning homework assignments to each TA. You should keep processing TA queries until you reach the end of file. The entries in each line of the TA queries file are separated by a single space, and each query is on a line by itself. The file is guaranteed to be well formed.

**Program Design and Classes**

For convenience, here is a list of the classes involved in this homework assignment. There are a total of five classes in this program. Two of the classes are already developed and you have to write the other three.

1. `Name`: A Name object represents the name of a student submitting a homework assignment. The Name class must implement the `Comparable<Name>` interface. The Name objects are compared base on the above criteria: last name using String comparison, then break a tie using first name if the last names are identical. You should implement the Name class for this assignment.

2. `Files`: A Files object represents a list of files to grade. The Files class is provided for you. You should use this Files class to represent the files to grade for a given Homework assignment. You should **not** implement the Files class, or change its implementation. You must use the Files class as provided.

3. `Homework`: A homework assignment to be allocated to a TA and graded. Each `Homework` object should contain all the information on a single homework assignment. In particular, it should have a Name object to represent the author of the homework and a `Files` object to represent his/her submitted files. The `Homework` class must implement the `Comparable<Homework>` interface. As such, it must have a `compareTo` method that takes in another Homework. This `.compareTo` method should satisfy the standard `compareTo` contract and impose a natural ordering based on the priority of Homeworks. That is to say, if Homework x has higher priority than Homework y (i.e. x should come before y), then `x.compareTo(y)` should return a negative number. If x has lower priority than y, then `x.compareTo(y)` should return a positive number. If x and y have equal priority, then `x.compareTo(y)` should return 0. The natural ordering of Homeworks, implemented correctly, should rank Homeworks with higher priority ***before*** Homeworks with lower priority. Thus, in a sorted array the Homework with highest priority comes first. Another important method is the `toString` method. The `toString` method should return a String describing the homework using the format described in the javadoc. It should **not** print that String to the screen directly. You should implement the `Homework` class for this assignment.

4. `HomeworkQueue`: The `HomeworkQueue` is used for queueing up Homeworks based on their ordering (natural ordering). With a `HomeworkQueue` object, you can add Homeworks to the queue, or retrieve (and remove) the `Homework` with highest priority among every Homework in the queue. It is important to note that this "order of priority" is only as good as the `compareTo` method that you write in the `Homework` class. The `HomeworkQueue` class is provided to you with the assignment. You should **not** implement the `HomeworkQueue` class, or change its implementation. You must use the `HomeworkQueue` class as provided.

5. `Allocation`: The `Allocation` class is where your main method will reside. It should read in data from `HW_List.in` and `TA_Queries.in` and output to `HW_Allocation.out`. You should implement the `Allocation` class.

**General Notes**

All input for this program is from File. You should not prompt the user for any input or read from standard input.

All output for this program is to File. Anything you print to standard output or standard error will not be graded. You may use it for debugging if you like, however.

Do not use packages for this program. Simply keep all source files in the same directory.

## Deliverables

You must submit the 3 source code files (`Homework.java`, `Name.java`, `Allocation.java`) for your program over Webcourses as a separate attachment (i.e. do *not* create an archive/zip of the files). **In particular, *do not* submit any .class files!!! This will result in 0 credit for the assignment.** You must submit your source files as an attachment using the "Add Attachments" button. Assignments that are typed into the submission box will not be accepted. Please do not zip your files, and simply submit each file

## Documentation

Your code must include:

1. Header comments for each source file, with the following information: your name, course number, section number, and date.
2. Description of the functionalities and responsibilities of each class.
3. Comments for each public method and field in your classes.
4. Inline comments throughout the code explaining its logic and design.

## Restrictions

1. Your program must compile using Java 6.0 or later.
2. For each of the three classes you write, you must implement all the methods listed in the javadoc provided, even if a method is not necessary for the assignment problem.
3. Even if your program produces the correct output and match the reference output exactly, you will incur a significant penalty if your code does not implement the required classes with the required methods.

## Execution and Grading Instructions

1. First, create a directory and put in it source files `HomeworkQueue.java` and `Files.java`.
2. For each student, download student's source codes `Homework.java`, `Name.java and Allocation.java` into the source directory.
3. Review each of the source files for comments and appropriate methods.
4. Copy the judge version of `HW_List.in` and `TA_Queries.in` into the directory, and run the program using the command `java Allocation`. Compare output `HW_Allocation.out` with `HW_Allocation.judge.out` (the correct output) using fc or diff (depending on your OS).
5. If there are no differences, give full execution score. Otherwise, take off points according to the criteria provided (only to the TAs).

## Points Breakdown

Points Total: 140

- *Execution*: 80
- *Code*: 40
- *Documentation*: 20