Programming II   Final Exam Project
Due: June 17th (Saturday) by 23:59 to Final folder in LMS

You will implement the following project as your final. Please follow the instructions in the order given below. First important notes and a short project description.

Notes:

The project should be completed individually. Any help from a friend, copy and paste from the internet or unreasonable similarity between codes of students would earn you no credit for the exam and disciplinary action taken.
The project should be implemented in exactly the same way described below without skipping steps. You can only use what you learned in the class (there should be NO MATERIAL in your implementation that I did not cover in class)
Your source code with your name, the output of your runs and comments should be turned into LMS "final" folder as a single pdf file.
There will be an interview with every student about his/her submission. Your grade for the final will be based on your answers to the questions. Note that you could be asked to write programs related to the topics of final during the interview. The interview will be held after the final exams are over. But if you finish it earlier than the due date, please let me know so that we can have your interview earlier. Failure to attend the interview will result in no grade for the final exam.

Project Description:
The final exam project is about different objects communicating with each other through (using) a central single object. That is, there are some number of objects that send messages to each other by asking the central object. Each object (communicator) will also save all messages sent and received during its lifetime in a file in the text based form. The messages are to be sent in a specific format that need to be parsed by the receiving communicator. A communicator can send a message to a single specific communicator or to all communicators. All this is supposed to be specified in the message itself. Every communicator should have a unique id (or name) associated with it and every communicator should register itself with the single central object once it is created. When a communicator does not want to communicate with others any longer, it should unregister itself from the single central object.
The central object is responsible for three things:

1) When a new communicator is created (new Car(...) for example) it will register itself with the central object (the central object adding the new communicator to its collection).
2) The central object after adding this new communicator to its collection of communicators, it will let all other communicators know about this new communicator by sending the unique identifier to all other communicators.
3) When a communicator wants to send a message to a single communicator, it will provide the source (the sender) the the destination identifier of the communicator with the message. And the central object is responsible for making it happen.

Now the order you need to implement the project:

Steps you should follow in your implementation (Make sure not to skip any step):

1. When a communicator is created they need to be assigned (probably in their constructor) a unique identifier or name. This name/identifier is saved with the center object. Design classes called Cat, Car, Cloud and Cup classes for communicator types. These classes should not be placed in any hierarchical structures (no extends relationship between them or no super class that these classes extends). Let's say you have two Car, three Cup objects created. You probably want all these objects to have a sequential unique id. Hint: Create a single class that generates the id's. That is, every time you create a new communicator, the about-to-be-created communicator goes and asks the id generator to provide a unique identifier. Is this id generator a class with a static method or an object with a non-static method that generates id's? (15 points)
2. The Center class should have a "collection" of these communicators. Once a communicator object is created, it should report (register) with the single center object. That is, registration makes sure that this communicator object reference is saved with the Center object. The Center object's collection will have references of objects of type Car, Cat, Cup and Cloud. How can you assign different types of object references into a single type where these objects are not related by inheritance (extends)? This "collection" of communicators could be a map or a list of communicators. When a communicator is created (new Car(...)) and it registers itself with the Center object, the center object first adds this new communicator object to its collection of communicators and sends a message to all communicators registered so far about this new object (the message sent is the unique identifier of the communicators just registered). The register or addMe method should be able to receive any object reference of type Car, Cat, Cup or Cloud. (45 points)
3. The message sent by a sender (communicator to a single or all communicators) has the following format:
   "srcID:destID:message" where the actual message has other pieces of information, specifically, the sender's id, the receiver's id and the actual message. The central object is going to receive the message and parse it,get the 3 pieces of it and send the actual message to the receiver (remember that the object whose id is 3 who is supposed to receive the message can only receive the message if the central object has an association of that id with the receiver's reference. "destID" could be "all" instead of a communicator id (which means all communicators as opposed to a single one). if dest is a list of ids separated by commas, then the sender wants to send it to those communicators with those ids only. "message" part could be "quote" if a communicator wants to send a randomly picked message from its 2D array. (65 points)
4. Once the sender sends the message, it will save the message to a file that will be used as a logger (all the messages sent or received will be saved in this file). Also, once the receiver receives the message, it will save the message to the file that belongs to the receiver. For every communicator there is a unique file (text based file) that keeps track of (saves) all the messages sent and received. If a communicator wants to send a message to every other communicator (of course except for itself), the destination id should be "all" in the message. (75 points)
5. Every communicator will have a two-dimensional array with 2 rows and 2 columns of type String. That is, four cells. Each cell will contain a quote for the day. For example, "programming is beautiful", "love what you are doing", "every single day is a vacation for me", "(because) I love what I am doing". If the last part of a message

is “quote”, then the sender will pick one of the four quotes randomly and send it as its message. (85 points)

6. Write a report about half a page or a page that explains how you solved the steps. This report should be appended to your .java file as comments. Make sure to mention also what data structures you chose and why. (100 points)

Methodology you should follow:
When you are designing a system (of objects and their relationship to each other), always start from a simple model (system). For example, first create communicator objects who can get a unique id. As you can see, in a sense, I close my eyes for the rest of the project and solve a simple problem (a sub problem that I need to solve for the overall problem). This simple simple step is Step 1 in the project description. That is, Step 1 is saying that your entire project is to be able to create objects with unique ids. Test your Step 1 by generating a unique id for every object (which are going to be your communicators) and print them.
Next step, can I design a Center object (a single object) that can “collect” every communicator that is created.

Deliverables:

Your source code with your name, the output of your runs and comments should be turned into LMS “final” folder as a single pdf file.