344-715 Project 1 – Due Wed. April 13th
Visit to the Museum

There are *m* passengers, *n* cars, and *c* controllers; Passengers wander around the museum for a while, then line up to take a ride in a car (have each passenger block on a different object)**.**
Each car has *s seats*.
When a car is available (empty), it loads the passengers (loading the passengers must be done in a FCFS order, similar to the way done in rwcv.java);
Once the car is full (In your implementation the passengers that get on the same car, must now block on the same object)**, it asks** (signals and then waits for) **one of the controllers' permission for departure.**
Note: If a passenger doesn't get the chance to get in the car, it will wait for the next car available. Keep in mind that the last car might not be full.
The controller checks the tickets (sleep of random time) **and next allows the departure of the specific car.**
Once the permission is given, the car rides around the park for a random amount of time. After the ride ends, the car unloads the passengers and becomes available again.
After all passengers have the chance to take a ride, the threads can terminate.

Synchronize the *m* passenger threads, the *n* car threads and the *c* controller threads based on the given story.

Closely follow the story and the given implementation details.
Besides what is already mentioned, make sure that when a thread must wait, it cannot do busy waiting, it must block through wait.
Mutual exclusion implementation must also be implemented through synchronized methods and/or blocks.

The number of passengers – m, cars – n and controllers –c and the initial number of available seats – s should be entered from the command line.

Default Values:	numPassengers m = 23
	numCars n = 3
	numControllers c = 2
	numSeats s = 5

Do not submit any code that does not compile and run. If there are parts of the code that contain bugs, comment it out and leave the code in. A program that does not compile nor run will not be graded.

Follow the story closely and cover the requirements of the project's description. Besides the synchronization details provided there are other synchronization aspects that need to be covered. You can use synchronized methods or additional synchronized blocks. Do NOT use busy waiting. If a thread needs to wait, it must wait on an object (class object or notification object).

3. Main class is run by the main thread. The other threads must be manually specified by either implementing the Runnable interface or extending the Thread class. Separate the classes into separate files. Do not leave all the classes in one file. Create a class for each type of thread.

Add the following lines to all the threads you make:

```
   public static long time = System.currentTimeMillis();
   public void msg(String m) {
       System.out.println("["+(System.currentTimeMillis()-time)+"] "+getName()+": "+m);
   }
```

There should be printout messages indicating the execution interleaving. Whenever you want to print something from that thread use: msg("some message here");

NAME YOUR THREADS or the above lines that were added would mean nothing. Here's how the constructors could look like (you may use any variant of this as long as each thread is unique and distinguishable):

```
// Default constructor
public RandomThread(int id) {
    setName("RandomThread-" + id);
}
```

Design an OOP program. All thread-related tasks must be specified in their respective classes, no class body should be empty.

DO NOT USE System.exit(0); the threads are supposed to terminate naturally by running to the end of their run methods.

Javadoc is not required. Proper basic commenting explaining the flow of program, self-explanatory variable names, correct whitespace and indentations are required.

<u>Tips:</u>
-If you run into some synchronization issue(s), and don't know which thread or threads are causing it, press F11 which will run the program in debug mode. You will clearly see the thread names in the debug perspective.

<u>Setting up project/Submission:</u>

In Eclipse:
Name your project as follows: LASTNAME_FIRSTNAME_CSXXX_PY
where LASTNAME is your last name, FIRSTNAME is your first name, XXX is your course, and Y is the current project number.

For example: Fluture_Simina_CS344-715_p1

To submit:
-Right click on your project and click export.
-Click on General (expand it)
-Select Archive File
-Select your project (make sure that .classpath and .project are also selected)
-Click Browse, select where you want to save it to and name it as LASTNAME_FIRSTNAME_CSXXX_PY
-Select Save in **zip format**, Create directory structure for files and also Compress the contents of the file should be checked.
-Press Finish

Email the archive with the specific heading: **CS344-715 Project # Submission: Last Name, First Name** to simina.fluture@gmail.com
You should receive an acknowledgement within 24 hours.