CSCI 340, Fall 2018
Instructor: Simina Fluture, PhD
Project 1 – **Due date: Dec. 12 with no penalty until Dec 16 (submissions after 16 are not considered for grading)**

Java – Code Implementation

Using Java programming, synchronize the threads, in the context of the problem. Closely follow the implementation requirements. The synchronization should be implemented through Java semaphores and operations on semaphores (acquire and release)

For Mutual Exclusion implementation use **Mutex semaphores**, no volatile variables.

For semaphore constructors, use ONLY: Semaphore(int permits)
Creates a Semaphore with the given number of permits and nonfair fairness setting.

As methods use ONLY: acquire(), release(); You can also use:

getQueueLength()
Returns an estimate of the number of threads waiting to acquire.

hasQueuedThreads()
Queries whether any threads are waiting to acquire.

DO NOT USE ANY OF THE OTHER METHODS of the semaphore's class, besides the ones mentioned above.

Any wait must be implemented using P(semaphores) (acquire).

Any shared variable must be protected by a mutex semaphore such that Mutual Exclusion is implemented.

Document your project and explain the purpose and the initialization of each semaphore.

DO NOT use synchronized methods (beside the operations on semaphores).

Do NOT use wait( ), notify( ) or notifyAll( ) as monitor methods. Whenever a synchronization issue can be resolved use semaphores and not a different type of implementation.

You should keep the concurrency of the threads as high as possible, however the access to shared structures has to be done in a Mutual Exclusive fashion, using a mutex semaphore.

Many of the activities can be simulated using the sleep(of a random time) method.

Use appropriate System.out.println( ) statements to reflect the time of each particular action done by a specific thread. This is necessary for us to observe how the synchronization is working.

Submission similar to project1. Name your project : **Doe_John_CS340_p2**
Upload it on Blackboard.

CSCI 340, Fall 2018
Instructor: Simina Fluture, PhD
Project 1 – **Due date: Dec. 12 with no penalty until Dec 16 (submissions after 16 are not considered for grading)**

A day of "fishy" school

Small fish are going to school every day. They first arrive at a meeting place (simulate fish getting to the meeting place by using **sleep(random_time)**), where they group in, GS, group size (use semaphores). From there, they will use a manta ray, named Mantis, as way of transportation to get to the school at a remote reef. Mantis waits until all the fish are gathered together (use semaphores). The last fish to group will let Mantis know that they are ready to move on. (use semaphores).

Mantis, whose capacity is MC takes as many groups as it can and transports them to the classroom at the remote reef. It will signal the members of the specific groups (use semaphores). Mantis comes back as many times as needed to transport all the fish groups. (you need to find a way of showing what fish is in each group and what group(s) are currently transported). Once Mantis get to the school, it will signal the fish he just transported that they arrived (use semaphores). Next fish will wait for the school day to end (use semaphores).

When Mantis is done transporting fish to school, he will take a break (sleep of a longer time).
When Mantis wakes up he will signal one of the fish (use semaphores) and wait for all the fish to leave (use semaphores). That fish will signal another fish (use semaphores and platoon policy). The last fish to leave will let Mantis know that he can leave as well.

Using the synchronization tools and techniques learned in class, synchronize the fish thread and Mantis threads – in the context of the problem described above.
The number of fish should be entered as an argument.
In order to simulate different actions you must pick reasonable intervals of random time. Make sure that the execution of the entire program is somewhere between 40 seconds and 90 seconds.

Default group size GS=3 Default fish number FN=13 Default Mantis capacity MC=7

CSCI 340, Fall 2018
Instructor: Simina Fluture, PhD
Project 1 – **Due date: Dec. 12 with no penalty until Dec 16 (submissions after 16 are not considered for grading)**

Additional Guidelines

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.
Closely follow all the requirements of the Project's description.

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. Don't create packages.**

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);
   }
```

It is recommended to initialize time at the beginning of the main method, so that it will be unique to all threads.

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

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 its respective classes, no class body should be empty.

Setting up project/Submission:

In Eclipse:

Name your project as follows: LASTNAME_FIRSTNAME_CSXXX_PY

CSCI 340, Fall 2018
Instructor: Simina Fluture, PhD
Project 1 – **Due date: Dec. 12 with no penalty until Dec 16 (submissions after 16 are not considered for grading)**

where LASTNAME is your last name, FIRSTNAME is your first name, XXX is your course, and Y is the current project number.
For example: Doe_John_CS340_p2

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 (.zip)**

-Press Finish

PLEASE UPLOAD YOUR FILE ON BLACKBOARD ON THE CORRESPONDING COLUMN.

The project must be done individually with no use of other sources including Internet. No plagiarism, No cheating.