**344- 715 SP2018**
**DUE DATE: April 20**

# The Shuttle-Station Monitor

Our space fleet contains **numShuttle** shuttles. As an initial state, all shuttles are in the air, cruising (simulated by sleep of random time), and the space station is empty.

Once a shuttle lands, the shuttle will first recharge. It will move to the recharging station (sleep of random type). There are **numRecharge** spaces in the recharging station. Shuttles are **waiting** in groups of size numRecharge for the controller to allow them to proceed (use a different notification object for each group).

The controller, opens the station every half hour. Once the recharging station is open, the controller will signal the first group of shuttles to proceed. Each shuttle needs a random % of its tank capacity (for example 25%, or 70%. Generate a random number between 50 and 100, representing the percentage of needed fuel) to fill up its tank. The capacity of the station's reservoir is 200%. If in the reservoir there is not enough fuel to fill a shuttle's tank, the controller will be signaled for assistance. The controller will refill the reservoir and a signal will be sent to the shuttle that was waiting to be filled-up. The last shuttle in the group to be done charging will signal the controller.

Next, the shuttles move to the landing/takeoff area (sleep of random type) waiting for the next take off (each shuttle blocks on a different object in a FCFS order. Use an implementation similar to the one on rwcv.java). When it is time for takeoff, the shuttles will be signaled one by one by the aircraft supervisor. There are only **three** take-off tracks.

Once a shuttle takes off, it will start cruising again and everything gets repeated.
At the end of the day, all the threads should terminate. Set the length of the day in such a way that each shuttle had at least 3 cruises.

They are ***numShuttles*** shuttles, one controller and one supervisor.

Initial values: numShuttles = 7
numRecharge = 3

The numShuttles should be an argument of the program.

***Develop a monitor that will synchronize the threads: shuttle, controller, and aircraft supervisor in the context of the problem.***

**Closely follow the story and the given implementation details.**

*Choose the appropriate amount of time(s) that will agree with the content of the story. I haven't written the code for this project yet, but from the experience of grading previous semester's projects, a project should take somewhere between 45 seconds and at most 1 minute and ½, to run and complete.*

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 to make sure that mutual exclusion over shared variables is satisfied. 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.

Tips:
-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.

**Setting up project/Submission:**

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

**Upload the project on BlackBoard.**