**CSCI 340**
**Project 1 Fall 2023**
**DUE: Tuesday, November 21 (end of day)**
**Wednesday, November 22 (end of day – 7 pts penalty)**
**After November 22 no submission will be considered for grading**

The project must be done individually - no exceptions. Plagiarism is not accepted. Just changing the name of variables and the methods will not make the project yours. Receiving and/or giving answers and code from/to other students enrolled in this or a previous semester, or from a third-party source including the Internet is academic dishonesty subject to standard University policies.

The objective of this project is to familiarize you with the creation and execution of threads using the Thread class methods. You should use, when necessary, the following methods to synchronize all threads: **run(), start(), currentThread(), getName(), join(), yield(), sleep(random time), isAlive(), getPriority(), setPriority(), interrupt().**

The use of semaphores to synchronize threads is strictly DISALLOWED. Additionally, you are NOT PERMITTED to use any of the following: wait(), notify(), notifyAll(), the synchronized keyword (for methods or blocks), and any synchronized collections or synchronization tools that were not discussed in class or mentioned here.

You CAN, however, use the modifier **volatile** and the **AtomicInteger** and **AtomicBoolean** classes if you choose to.

**Directions:** Synchronize the *customer*, *cashier, and adoption clerk* type of threads in the context of the problem described below. **Please refer to and read carefully the project notes, tips and guidelines before starting the project.**

| | | |
|---|---|---|
| Thread types: | customer | (num_customers threads, default 20) |
| | cashier | (num_cashiers threads, default 3) |
| | adoption_clerk | (one) |

Other variables' default values: num_visitors = 3; num_pets = 12

## Pet-Adoption Event

The Happy Pet store holds a pet-adoption event.
Customers commute to the pet store in different ways (simulated by sleep of random time). Once arrived at the store, the customer will generate a random number between 1 and 10 inclusively. If the number is <4, the customer will only buy food and toys for his/her pets. Otherwise, if the number is even, the customer will be interested in adopting a pet only, else the customer will first do some shopping and next he/she will also check the pets that are for adoption, maybe they will adopt one.

If a customer will buy food and/or toys he/she will rush to get it done. (*Use* ***getPriority()****,* ***setPriority()****, and sleep(random time); after the customer has increased its priority, it will sleep for a random time. As soon as it wakes up make sure you reset its priority back to its normal value)*. They will browse the aisles (*sleep of random time*) and pick what is needed. Next, they

will (busy) wait at the cashier.  There is one line only but three cashier-clerks.  The cashier-clerks assist the customers in a FCFS order.
Note: *The line can be simulated using java.util.Vector.  Since Vector is already synchronized, we do not need to synchronize when removing/inserting elements.*

After shopping if the customer is not interested in seeing the pets to be adopted, he she will leave.

A customer who wants to check the pets (no matter if he/she shopped or not), will wait (*simulated by a sleep of long time*) to be allowed to enter the area where the pets are available to be seen.  In order to avoid congestion and accidents, only *num_visitors* are allowed in that area.   Whenever there is an available space, the *adoption_clerk* will announce a waiting customer in a FCFS order, by interrupting the customer (*use the **interrupt()** method. In the body of the catch have a message that will show that a specific customer has been interrupted). Note: in the adoption area there will also be customers who came solely to adopt a pet.*

Once announced the customer will check all the pets in the room (*simulated by sleep of random time*) Deciding to adopt or not will be done randomly (*generate a random number between 1 and 10. If < 6 then adopt*).   If they decide to adopt a pet, the adoption clerk will update the number of available pets. If all pets will end up being adopted, then no more visitors will be allowed in the adoption room;

Next the customer will leave the adoption room.  The ones who adopted will have to complete a few forms (simulated sleep of random time).  Before doing that, they will take a break at the coffee center. (*use **yield()** twice*).  Once finished with all the formalities, the adopting customers are ready to leave with their new pets.  They will leave in decreasing order.
Let's say that customers 2, 4, 7, 11, 15, 17 adopted a pet.   If ready, customer 17 will leave first.  Next customer15 will leave and so on (use **join(), isAlive()**).

After all customers left the store, the cashiers will leave.  The last cashier to leave will let the adoption clerk know that it is time to leave as well.

To reiterate, you are not permitted to use monitors, semaphores, collections or any other synchronization tool if they were not discussed in class. You can, however, use the **volatile** modifier and the classes **AtomicInteger** and **AtomicBoolean**.

**Guidelines:**
1. 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.

2. Closely follow all the requirements of the project's description.

3. The main method is contained in the main thread. All other thread classes must be manually created 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). DO NOT create packages.**
4. The project asks that you create different types of threads.  **No manual specification of each thread's activity is allowed (e.g. no Student5.goThroughTheDoor())**

5. **Add the following lines to all the threads you instantiate:**

```
    public static long time = System.currentTimeMillis();

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

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

6. There should be output messages that describe how the threads are executing. Whenever you want to print something from a thread use: msg("some message about what action is simulated");

7. NAME YOUR THREADS. 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);
}
```

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

9. thread.sleep() is **not busy wait**.   while (expr) {..} is busy wait.
    Interrupting a thread should be done **ONLY** on the simulation mentioned in the project and not as a way of substituting busy waiting.
A thread that is sleeping for a long time should be released by an interrupt and **not** by using the method isInterrupted().

10. FCFS should be implemented in a queue or other data structure.

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

14.  Javadoc is not required. Proper basic commenting explaining the flow of the program, self-explanatory variable names, correct whitespace and indentations are required.  Headers with information about class name, last modified time, author… is mandatory for each java file.

**15.** 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 semesters' projects, a project should take somewhere between 10 and 25 seconds to run and complete.  Set the time in such a way so you don't create only exceptional situations.

**A helpful tip:**
-If you run into some synchronization issues, and don't know which thread or threads

are causing it, press F11 (in Eclipse) which will run the program in debug mode. You will clearly see the thread names in the debug perspective.

**Setting up project/Submission:**

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: Doe_John_CS340_p1

Archive all your source files (NOT CLASS FILES) in a .zip format (**NOT .rar**)

PLEASE UPLOAD YOUR FILE TO BLACKBOARD ONTO THE CORRESPONDING COLUMN.

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