CSCI 340
Project 2 Fall 2023

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.

You have the following choices:
1. You can submit a pseudo-code implementation:
Due Date: Saturday, December 9; end of the day (no late submission allowed)
OR
2. You can submit a java-code implementation;
Due Date: Tuesday, December 12; 4:00PM (no late submission allowed)
DO NOT SUBMIT BOTH TYPES OF IMPLEMENTATION: EITHER YOU SUBMIT THE PSEUDO-CODE, OR YOU SUBMIT THE JAVA-CODE.
Directions: You are asked to synchronize the threads of the following story using semaphores and operations on semaphores. Please refer to and read the project notes carefully, tips and guidelines before starting the project.
Plagiarism is not accepted. Receiving and/or giving answers and code from/to other students enrolled in this or a previous semester, or a third source including the Internet is academic dishonesty subject to standard University policies.

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 (s)he will browse the aisles (*sleep of random time*) and pick what is needed. Next, they will **wait** at the cashier. There is one line only but three cashier-clerks. The cashier-clerks assist the customers.
Note: for pseudo-code the sem queue is as default FCFS. For java-code implementation there is not a required order since a random thread from the sem queue will be released.

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 (s)he shopped or not), will **wait** 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 signal a waiting customer.
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 semaphores*).

After all customers have 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. (*use semaphores*)

1. Pseudo-code implementation

You are asked to synchronize the threads of the story using semaphores and operations on semaphores. Do NOT use busy waiting.
Any **wait** must be implemented using the P(semaphore).
Any shared variable must be protected by a mutex semaphore such that Mutual Exclusion is implemented.

Use pseudo-code similar to the one used in class (NOT java pseudo-code).
Mention the operations that can be simulated by a fixed or random sleep_time. Mention possible data structures that you might use in your implementation.
Your documentation should be clear and extensive.
In the documentation, give the type and initial value for each semaphore. Give the reason for each semaphore that you used in your implementation (specify who will do P and who will do V on that semaphore).

Deadlock is not allowed. For the submission you can have everything written in **one** (docx, pdf or txt file) or you can have different files (in the mentioned types) for each class. The files' names should contain your full name. They don't have to be zipped.

2. Java-code implementation

Do NOT use busy waiting, interrupts or any other synchronization tools besides the methods of the semaphore class specified below

The synchronization should be implemented using the Java Semaphores class and methods on semaphores (acquire and release)

Keep in mind that semaphores are shared variables (most of them should be declared as static)

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

In methods use ONLY: acquire(), release();

You can also use: getQueueLength() It returns an estimate of the number of threads waiting to acquire.

DO NOT use synchronized methods or blocks, Do NOT use wait(), notify() or notifyAll() as monitor methods. Whenever a synchronization issue needs to be be resolved use semaphores and not a different type of implementation (no collections, no more atomic classes)

Any **wait** must be implemented using P(semaphores) (acquire).
If shared variables are used, they 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.

The purpose of the of project is to evaluate your knowledge in using semaphores as synchronization tool and not your capability of submitting a "make believe" output.

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: YourLastname_Firstname_CS340_p2
Upload it on Blackboard.

- 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.
- 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.
- Each class must contain a header.
- 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 that you initialize the time at the beginning of the main method, so that it is unique to all threads.

  NAME YOUR THREADS. Design an OOP program. All thread-related tasks should be specified in their respective classes, no class body and run method 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 the program, self-explanatory variable names, correct whitespace and indentations are required.

Setting up (java) 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 P is the current project number.

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