CSCI 340, Fall 19
Instructor: Simina Fluture, PhD
Project 1 – **Due date: November 27th**

The project must be done individually. No exceptions.
Through its implementation, this project will familiarize you with the creation and execution of threads, and with the use of the Thread class methods. **In** order to synchronize the threads you will have to use (when necessary), run( ), start( ), currentThread( ), getName( ), join( ), yield( ), sleep(time), isAlive( ), getPriority( ), setPriority( ), interrupt( ), isInterrupted( ), and maybe synchronized methods.

In synchronizing threads, DO NOT use any semaphores. DO NOT use wait( ), notify( ) or notifyAll();

# Shopping at the Mega store

A customer walks into a Mega showroom and browses around the store for a while ((*simulated by **sleep(random time)***)) before (s)he finds an item (s)he likes. When a customer does so, (s)he might be a bit undecided if (s)he should buy or not purchase the item (simulate this with ***yield()*** *twice*). Once decided, the customer must go to the floor-clerks and receive a slip with which (s)he can pay for the item and then take it home.

Floor clerks wait (by doing ***busy waiting***) for customers to arrive; then, they help them (*in a FCFS order*) with whatever information they need. However, floor clerks can only help one customer at a time; therefore, a customer must wait (*busy waiting on a shared variable*) for an available clerk to help him/her.

After all of the customers are assisted (*you should keep track of the number of assisted customers*), the floor clerks wait for closing time (***sleep(of a long time)***).

Once a customer gets the information needed from the floor clerk, (s)he will go to the cashier to pay for the item. To do so, (s)he will increase his/her priority (use ***getPriority( ), sleep( of short time), setPriority( )***). Once the customer is at the cashier, the customer will reset his/her priority back to the default value.

There are two cashiers. One takes only cash, the other only credit cards. The customer will decide in which way (s)he will pay, cash or credit (determined randomly) and will get on the corresponding line.

After the item is paid for, the customer will take a break in the cafeteria and let the rest of the customers do their shopping. When the shopping is done, each customer will join another customer; they will leave in sequential order. Customer *N* **join**s with customer N-1, customer N-1 joins with N-2, …, customer 2 joins with customer 1 (use **join( )**, **isAlive( )**). Customer 1 will not join with any other customer; before (s)he leaves. Customer1 will announce to the floor-clerks that it is closing time by waking them up (call *interrupt()* on these threads).

The cashiers wait for customers and serve the customers waiting on their line in a **first-come first-serve basis.**
They will also terminate at the closing time.

---------------------------------------------------------------------------------------------------------

CSCI 340, Fall 19
Instructor: Simina Fluture, PhD
Project 1 – **Due date: November 27th**

Using the synchronization tools and techniques learned in class, synchronize the customer and floor clerk threads in the context of the problem described above.

Your program should have **three types of threads**:
**Customer threads**
**Floor-clerk threads**
**Cashier threads**

The number of customers should be read as command line arguments.
Default values are: number customer: 12
number floor-clerks: 3
number cashiers: 2

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.

## <u>Guidelines</u>

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

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

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. Don't create packages.**

4. The program asks you to create different types of threads. There is more than one instance of the thread. **No manual specification of each thread's activity is allowed (e.g. no Customer5.goOverPassage())**

**5. 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.

6. 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");

7. 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);
}
```

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

9. **No** implementation of semaphores or use of **wait( ), notify( ) or notifyAll( )** are allowed.

10. thread.sleep() is not busy wait.   while (expr) {..} is busy wait.

11. "Synchronized" is not a FCFS implementation. The “Synchronized” keyword in Java allows a lock on the method, any thread that accesses the lock first will control that block of code; it is used to enforce mutual exclusion on the critical section.  **FCFS should be implemented in a queue or other data structure**.

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

13.  A command line argument must be implemented to allow changes to the **nCustomer** variable.

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

Tips:
-If you run into some synchronization issues, 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.

CSCI 340, Fall 19
Instructor: Simina Fluture, PhD
Project 1 – **Due date: November 27th**

**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: Doe_John_CS340_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 (.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.**