Project 1 WN24
Due Date: Saturday, Jan. 13, 3:00PM
(not one minute later)

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's class methods. You should use, when necessary, the following methods to synchronize all threads: **run(), start(), currentThread(), getName(), isAlive(), 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 *athlete* and *judge* type of threads in the context of the problem described below. **Please refer to and read carefully the <u>project notes, tips and guidelines</u> before starting the project.**

Thread types:	**athlete**	**(num athletes / team, default 8)**
	judge	

The Competition

The athletes of two teams (the Green team and the Red team) will compete in a race with three obstacles: The Forest, The Lake and The Field.
Before the first obstacle and between any two obstacles, the athletes will stop for rest and/or food (*simulated by sleep of a random time*).

The obstacles are described below:

<u>The Forest:</u> *Use getPriority, setPriority, yield*
At the entry of the forest, the athlete will increase its priority, by a randomly generated number between 1-3. The forest will be a file of more than 300 but less than 600 randomly generated words of 4 letters chosen from the set {a, b, c, e}. The athlete using a compass will have to find in the forest a map that contains a magic word. The length of the magic word will also be 4 letters long, containing letters from the same set {a, b, c, e}. When the map with the magic word is found, the athlete can leave the forest immediately. Otherwise the athlete will have to search

the entire forest and by not finding the map in the forest he will be penalized (by having to yield( ) the CPU). After finishing this obstacle, the athlete resets its priority to default.

<u>The Lake:</u> *use sleep(of long time), interrupt()*
After resting, each athlete will line-up, ready to start the Lake obstacle.
Let's consider team Red with its athletes lined-up in the following order: R4, R5, R2, R1, R3, R7, R6
In the project you should display a listing of the line-up.
In this case R4 will proceed first, it will cross the lake forward-and-back (simulated by sleep of random time for each direction) while all the other participants are resting (simulated by a sleep of **long time**). Once back, R4 will **interrupt** R5. Next R5 will cross the lake forward-and-back and so on. The time of crossing the lake forward and back for each athlete should be displayed and recorded. The two teams will complete this obstacle concurrently.
In the body of the catch have a message that will show that a specific has been interrupted.
Have a basic *try and catch* (*InterruptedException e)*; don't throw them.

<u>The Field:</u> *use sleep(random time), counter, busy waiting*
This is an obstacle that will be completed next day. Athletes and the judge will meet in the cafeteria for breakfast. The last athlete to arrive at the cafeteria will let the judge (who was busy-waiting) know that they are all in for the next obstacle. The judge lets them know that the race will start in ½ hours and he will meet them all there (*have the judge sleep for 3000ms*).
The athletes will proceed to the athletic field ((*simulate this by having the athletes yield once, followed by random sleep between 1000ms and 5000ms*) and next busy-wait for the race to start.
The judge will announce the start of the race. Athletes who didn't make it there on time, will get disqualified and they will leave (terminate) right away. *Make sure you have a message from the disqualified athletes.* The remaining athletes will run and complete the race (*simulated by sleep of random time between 75ms and 200ms*). Their time should be displayed and recorded.

It is time for the awards. While the athletes busy wait for the results and awards to be posted and announced, the judge will compile all the points and times.
There will be two types of awards: individual by the obstacle and overall by the team.

<u>Forest</u>: athletes who found the word get the first prize.
The team earns a number of tokens equal to the number of winning athletes.

<u>Lake</u>: the athlete with the best time gets the first prize.
The team with the shortest cumulative time earns three tokens.

<u>Field:</u> the athlete with the best time gets the first prize.
The team with the lowest sum of the shortest two individual times earns two tokens.

In the end, the team with the largest number of tokens gets the first prize.

The judge will display a summary of all the results and awards.

Now it is time for everyone to leave. The winning team will have the privilege to leave first. The athletes of the second team will wait until the first team leaves (*use isAlive() and join()*)
The judge will leave after all the athletes are gone. (*use a counter*).

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 and Rules:

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 Athlet5.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 or getting out of a deadlock.
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… are 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.

<u>Setting up project/Submission:</u>
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.