Csci340 – Spring 2018
Faculty: Dr. Simina Fluture

The Race
CSCI 340 - Project 1

Due: Friday, April 27 (no late penalty until Sunday midnight)

The project must be done individually with no other source. No exceptions.
Read one more time the Academic Integrity link.

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();

In the next story they are two types of threads: racer and judge.

The Race

A number of racers will compete in a race with three obstacles: The Forest, The Mountain and The River. Before the first obstacle and between any two obstacles, the racers will stop for rest and food (by **sleeping for a random time**). The sequence of events will be something similar to:

Rest
Forest
Rest
Mountain
Rest
River
Get Home

Note: you will need to keep track of the time when a racer started a specific obstacle and when it ended that obstacle, together with the time when the racer started the race until he finished the race.

The obstacles are described below:

The Forest:
You will have to use the *setPriority( )*, *getPriority( )* and *yield( )* methods.

At the entry to the forest, the racer will try to rush as much as possible (simulated by increasing its current priority by a randomly generated number between 0-4).

The racer using a compass will have to find in the forest a map that contains a magic word. The forest will be a file of more than 300 but less than 500 randomly generated words of 4 letters chosen from the set {a, b, c, d}. The length of the magic word will also be 4 letters long, containing letters from the same set {a, b, c, d}. Immediately after the search the racer will reset its priority to the default value.
If the map with the magic word is found, the racer can leave the forest right away. Otherwise the racer will have to search the entire forest and by not finding the map in the forest will be penalized (by having to ***yield( )*** the CPU twice).

The Mountain:
The racers have to pass a very narrow passage. Only one racer can be on the passage at a given time. In the order of their arrival, the runners add themselves (get on line) on a shared vector. Each racer will have its own variable let's say *ready*. Ready is initialized to False. The last racer (vector size is equal to number of racers) sets the *ready* variable of the first racer to True. Each thread will **busy wait** until its *ready* variable becomes True. After a specific racer crosses the passage (sleep of random time), it will enable the next racer on line to cross it too.

The River:
Before the River obstacle, the racers will take a long rest (sleep of long time). They will be interrupted from sleep by the Judge. (use the ***isInterrupted( )*** and ***interrupt( )*** methods) Each racer will cross the river by sleeping for a random time.

Go Home:
Each racer knows the next racer in the sequence. For example, racer1 knows racer2, racer2 knows racer3 and so on (you will have to use the **isAlive( )** and **join( )** methods). After a particular racer finishes the last obstacle, it will check if its friend is still in the race by using the isAlive( ) method. If yes, then the racer will wait for its friend using the join( ) method, otherwise the racer goes straight home. Consider that the last created racer will just go home immediately.

Judge:
The Judge will interrupt the racers from their long rest before The River obstacle.

The judge is also responsible with presenting the order in which racers ended the race. There will be two reports:
1. First report will consider the time it took for each racer to end the entire race. (Including the resting time)
2. The second report will consider the times that it took for each racer to pass each of the obstacles. (Excluding the resting time)

Csci340 – Spring 2018
Faculty: Dr. Simina Fluture

Develop a multithreaded Java program simulating the Race operations.

Your program should have two types of threads:
Racer **threads (default number for racers is 10)**
Judge **threads**

The number of racers should be read as command line arguments. The default value is 10.

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.

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. 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 Racer1.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 nRacer **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.

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

Csci340 – Spring 2018
Faculty: Dr. Simina Fluture

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.