CSCI 340 Lab Project Writing Threads in java

This project is meant to give you a taste of coding 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( ), synchronized methods

The fishermen and the elusive "big catch"

Lake Big Fish is known for it's very large fish. There is a town on the shore called Breton. Near Breton is an island called Marrowind with a fishing hole where the large fish often come to feed. This fishing hole has become famous among amateur fishermen who come from all around to fish in the famous fishing hole. There is a shop in Breton that will stuff the prize and mount it for the fishermen so they can come home with proof of their "Big Catches". The fishermen can also bring in lesser fish for money. And use very small fish for bait. Every fisheman (**num_fm)** will leave once he has at least one "Big Catch" and enough money to pay for his trip, **vacation_cost**.

When a fisherman has a fish big enough to mount or cash in for money he takes a boat to the town and goes to the shop (use sleep() to simulate the boat trip). When he gets to the shop he sets his variable **needs_help(i)** to true and **busy waits** until a clerk is ready to help him. There are **num_ca**, customer associates but only one line of fishemen. A customer associate should pick the next waiting fisherman in a mutual exclusion fashion (this can be done from inside of a **synchronized** method) and in a FCFS order.

A fisherman that has not achieved his "Big Catch" or not recouped his **vacation_cost** will travel to the island to get his turn at the fishing hole. Fishermen gather on the island of Morrowind and **wait** (simulated by **sleeping** a long time) for the fishing hole to be available. The fishing hole is guarded by a Ranger. The ranger randomly interrupts one of the fishermen (use **interrupt()** and **isInterrupted()**). The ranger allows the fishermen access to the fishing hole and allowed one turn to fish. If the fisherman gets no fish, the Ranger allows the fisherman to increase his priority for a very short time and fish one more time. (use **getPriority( ), setPriority( )** and **currentThread( )** methods). After that time at the fishing hole the fisherman will immediately reset his priority back to the default value and will allow another fisherman to use the fishing hole (use **yield( )**). If the fisherman is successful he will receive a random fish (bait, 10, 20,30,40,50 or 200 pound fish) If the fisherman does not have at least 10 pounds of fish he must wait for the ranger to give him access to the fishing hole again. When he gets the 10 pounds of fish he can go back to the shop in Breton and get his mounted fish and or his money for the other fish ($0.75 per pound of fish)

One the fisherman achieves his goal of a big fish and **vacation_cost**. He will go home. However, each fisherman must join the previous fisherman in sequence. For example, in case that fisherman (i+1) is alive, fisherman (i) will join fisherman (i+1) (use **isAlive( )** and **join( )).** The last adventurer to leave for home (adventurer(0)) will let the Ranger know that it is time for him to terminate. The customer associates will terminate as well.

Using Java programming, synchronize the 3 types of threads, Fisherman, Customer associate and Ranger, in the context of the problem. Follow the requirements as specified.

The number of fishemen,(num_fm) , number of customer associates, and the vacation_cost must be entered as command line arguments. The default value for the number of fishermen is 6 and the vacation cost is $250 and number of customer associates is 2.

Guidelines:

1) Do not submit any project that doesn't compile and run
2) Follow the description in your code. I suggest putting comments in your code with the appropriate parts of the description.
3) The main or manager thread is run by the main procedure. All other threads are run by the main or manager thread. The other threads are created by either extending the Thread class or implementing the runnable interface. Each class should have its own file. Each type of thread is its own class. DO NOT CREATE PACKAGES.
4) Some of the threads have more than one instance. You should use an ArrayList to specify each thread.
5) Add the following line of code to your main or manager program"
   `public static long time = System.currentTimeMillis();`
6) method to all threads you make:

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

   This assumes the name of your main/manager class is BigCatchManager

7) There should be printout of messages indicating the execution interleaving. Also use the msg method to help in debugging
8) Name your threads in the constructor as follows. You may use variants of this as long as threads are unique and distinguishable:

```
public RandomThread(int id) {
        setName("RandomThread-" + id);
}
```

9) This should be designed as an OOP program, no classes should be empty.
10) You should not use semaphores (using wait(), notify() or notifyAll())
11) `Thread.sleep()` is not a busy wait. `while (expr) {….}` is a busy wait
12) "Synchronized" is not a FCFS implementation. The "synchronized" keyword in Java allows a lock on the method, any thread that access the method first has control over the method. It is used to enforce mutual exclusion on the critical section. FCFS should be implemented in a queue or other data structure.
13) Do NOT USE System.exit(). Threads are supposed to end naturally.
14) Command line argumentsmust be implemented to take values of: **num_fm num_ca, vacation_cost.**
15) Javadoc is not required. Proper basic commenting explaining the flow of the program, self-explanatory variable names, correct whitespace and indenting are required.

Setting up your project in Eclipse:

Name your Project LASTNAME_FIRSTNAME_CS340_P1.

To submit:

- Right click on your project and choose export
- Click on General (expand it)
- Select Archive File
- Select you project (make sure .classpath and .project are also selected)
- Select Browse and select where you want to store file. Name it LASTNAME_FIRSTNAME_CS340_P1
- Select save in zip format, **Create directory structure for files** and also **Compress the contents of the file** should be checked.
- Press finish
- Submit the archive on Blackboard.