Assignment 2 (1.5 points) - Deadline Aug 23 23:59 hrs

Learning goals

This assignment is about data sharing amongst threads and co-operative thread synchronization in Java. You are expected to learn the following skills by completing this assignment:

- Implement data sharing mechanisms among threads safely.
- Synchronize threads without wasting CPU cycles.
- Gain familiarity with java.util.concurrent library.
- Implement multi-threaded programs in Java and perform basic functional tests on these programs

Sensor Network

A *Sensor* is an object that measures humidity and temperature readings at certain time intervals and then sends that reading to one or many *Monitor*s. The *Monitor* computes the discomfort level based on the running averages computed from the sensor input. A *Subscriber* is an object that has subscribed to one or many *Monitor*s and is notified when subscribed discomfort levels are detected or exceeded. Each of these components can run independently in separate threads and interact as indicated in the diagram in Figure 1.

In order to compute the discomfort levels, use Table 1. The discomfort level for a pair of temperature and humidity readings is computed by looking up the discomfort levels for both readings in the table and returning the maximum. For example, if the temperature is 12 and the humidity is 75, the discomfort level is $3 = \max(1, 3)$.

Read the supplied interfaces of `assignment2.Sensor` and `assignment2.Monitor` and `assignment2.Subscriber` which need to be implemented in this assignment.

Temperature	Humidity	Discomfort Level
$(-\infty, 10)$	$(-\infty, 50)$	0
$[10, 20)$	$[50, 60)$	1
$[20, 30)$	$[60, 70)$	2
$[30, 40)$	$[70, 80)$	3
$[40, 50)$	$[80, 90)$	4
$[50, \infty)$	$[90, \infty)$	5

Table 1: Discomfort level lookup table

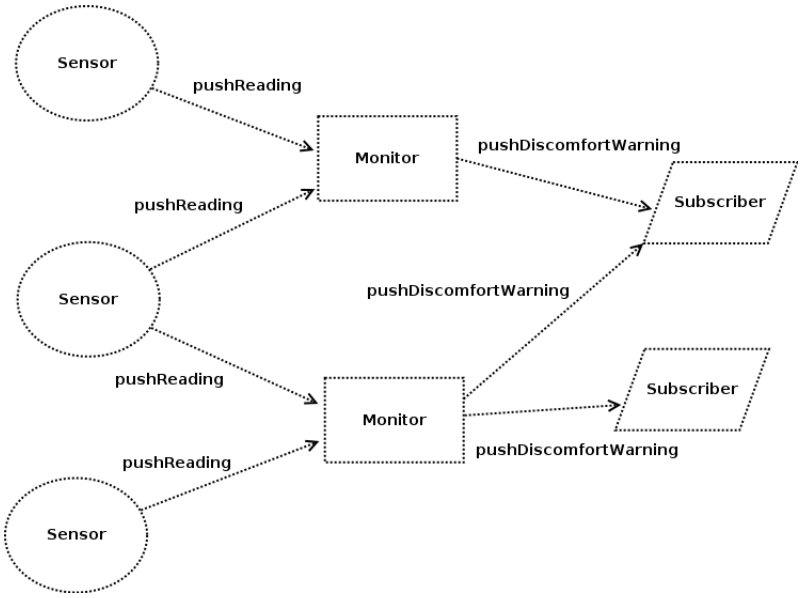

Figure 1: Sensor network architecture

Implementing the sensor network model

Implement the sensor

- The sensor continuously generates sensor readings and then pushes the readings to the Monitor using `assignment2.Monitor.pushReading` method.
- Implement the `assignment2.Sensor` interface by filling out the following stubbed out methods in `assignment2.ASensor` *AND* making them thread-safe.
    a. `generateSensorReading`: Generate a `SensorReading` object which represents a sensor reading. This method should not push readings. It is meant to be used as a sensor reading generator. For extra brownie points, generate probabilistically well spread out inputs.
    b. `registerMonitor`: Adds the list of monitors to which a sensor reading would be sent to.
    c. `run`: Runs the sensor by continuously performing a sensor reading and pushing it to the monitors. Change the given implementation to make it thread-safe.

Implement the Monitor

- A Monitor continuously reads the sensor readings pushed to it. It computes the discomfort level based on the running averages of the readings

and then pushes the discomfort level computed to the subscribers subscribed to the computed discomfort level or higher.

- Implement the `assignment2.Monitor` interface by filling out the following stubbed out methods in `assignment2.AMonitor` *AND* making them thread-safe.

    a. `pushReading`: Push the sensor reading from the sensor to the monitor. Can be blocking (e.g., by enqueueing to a bounded buffer) or non-blocking (e.g., by dropping readings if no buffer space is available). The latter non-blocking implementation is preferred. This method is invoked by the sensor thread to push sensor readings.

    b. `getSensorReading`: Pull the earliest pushed sensor reading from the available sensor readings that have been pushed by sensors. It should block if there are no readings available.

    c. `processReading`: Process the sensor reading by first computing the running average of the humidity and temperature and then computing the discomfort level of the running average. Once the discomfort level is computed, push the discomfort level reading to the subscribers subscribed for the same discomfort level or higher using `assignment2.Subscriber.pushDiscomfortWarning` method.

    d. `registerSubscriber`: Registers a subscriber for discomfort level events.

Implement the Subscriber

- A subscriber runs continuously to read the discomfort level readings pushed to it and processes them.

- Implement the `assignment2.Subscriber` interface by filling out the stubbed out methods in `assignment2.ASubscriber` *AND* making them thread-safe.

    a. `pushDiscomfortWarning`: Push the discomfort level reading from the monitor to the subscriber. Can be blocking (e.g., by enqueueing to a bounded buffer) or non-blocking (e.g., by dropping readings if no buffer space is available). The latter non-blocking implementation is preferred. This method is invoked by the Monitor thread.

    b. `getDiscomfortWarning`: Pull the oldest pushed discomfort level reading from the pushed readings. It should block if there are no readings available.

    c. `processDiscomfortWarning`: Process the reading by just printing it :-). You are free to be creative if you want.

Run the sensor network

- Implement the stubbed out *main* function in `assignment2.SensorNetworkHarness` class which creates multiple Sensor threads and multiple Subscriber threads and one or many (if you want to) Monitor threads and demonstrates the implemented interfaces. Make sure you test various possible interconnections among sensors, monitors and subscribers.