BackEnd: Take Home Exercise

Home interview questions

For the below set of questions, code as if it was going to be deployed to production. This means you should follow your usual concerns when writing production code – think about appropriate tests, maintainability, performance etc.

Think about when it is appropriate to use code you wrote in earlier questions.

You may use libraries from the internet where you think it's appropriate to do so, but not if it'd solve a core concern of the question.

Feel free to extend the interface where you think it'd benefit from doing so.

You are free to spend as much time on it as you want – you should at least do the first two questions. The latter two would be a further demonstration of your abilities. You should document how long each section took you – be honest.

=====

Implement the below interface so that we can pass in a List<NumAndProbability> - which denotes a set of numbers and the probability that they should be drawn when we call nextFromSample().

In the implementation, you may use a creational pattern / constructor / setter, but justify your choice.

```
public interface ProbabilisticRandomGen {


    public int nextFromSample();


    static class NumAndProbability {
        private final int number;
        private final float probabilityOfSample;


        public NumAndProbability(int number, float probabilityOfSample) {
            this.number = number;
            this.probabilityOfSample = probabilityOfSample;
        }


        public int getNumber() {
            return number;
        }


        public float getProbabilityOfSample() {
            return probabilityOfSample;
        }
    }
}
```

An event bus is similar to a messaging daemon – you publish messages to it, and subscribers receive those messages.

Write a version of the EventBus designed for single-threaded use (thread calling publishEvent is the same as the thread used for the callback on the subscriber).

Then write a version of the EventBus which supports multiple producers and multiple consumers (thread calling publishEvent is different to thread making the callback)

Do not use any non JDK libraries for the multithreading.

Extra points if you can extend the multithreaded version (maybe by extending the interface) so it supports event types where only the latest value matters (coalescing / conflation) – i.e. multiple market data snapshots arriving while the algo is processing an update.

```
public interface EventBus {



    // Feel free to replace Object with something more specific,
     // but be prepared to justify it
     void publishEvent(Object o);



    // How would you denote the subscriber?
     void addSubscriber(Class<?> clazz, ???);



    // Would you allow clients to filter the events they receive? How would the interface look like?
     void addSubscriberForFilteredEvents(????);



 }
```

A Throttler allows a client to restrict how many times something can happen in a given time period (for example we may not want to send more than a number of quotes to an exchange in a specific time period).

It should be possible to ask it if we can proceed with the throttled operation, as well as be notified by it.

Do not assume thread safety in the subscriber.

```
public interface Throttler {



    // check if we can proceed (poll)
     ThrottleResult shouldProceed();



    // subscribe to be told when we can proceed (Push)
     notifyWhenCanProceed(???);



    enum ThrottleResult {

        PROCEED, // publish, aggregate etc
         DO_NOT_PROCEED //
     }

}
```

Write a **general purpose** class that calculates statistics about a sequence of integers.

An example use case would be latency statistics – you can add latency measurements to it and get statistics out.

Some clients may only want to be notified when the statistics match certain criteria – e.g. the mean has gone above a threshold.

Do not assume that subscribers will be threadsafe. The add method may be called by a different thread than the callback should be made on.

```
public interface SlidingWindowStatistics {


    void add(?? measurement);


    // subscriber will have a callback that'll deliver a Statistics instance (push)
     void subscribeForStatistics(???);


    // get latest statistics (poll)
     Statistics getLatestStatistics();




    public interface Statistics {

        ?? getMean();

        ?? getMode();

        ?? getPctile(int pctile);

    }

}
```