Count requests received in last second, minute and hour

Problem: Given an extremely busy server which receives thousands of requests per second.
It is desired to find the number of requests received in the last second, minute and hour.
What algorithm and data-structures can be used to do this as accurately as possible?

Solution : Some of the first solutions that come to mind are:

1. Maintain 3 counters, one each for last hour, second and minute.
   This is quickly rejected because there is no way to remove the count of requests which fall out of the window of last hour, minute and second.

2. Maintain 3 lists, one each for last hour, second and minute
   This does solve the problem with the above counters, but since the number of requests is huge, a massive synchronization will be required for each thread adding its request to three lists.
   Given thousands of requests per second, synchronization alone will slow down the entire lists' updation process.

Best approach: A good solution is one which allows concurrent updates and does not eat up too much memory.
With this idea in mind, the following can be a good solution:

1) Create an array ***AtomicInteger frequencies[1000];*** to store the number of requests received per second.

2) This array will store frequencies of requests for the current second i.e. from HH:MM:SS to HH:MM:SS+1 time

3) We will store current second in some variable, say ***currentSecond***
This can be retrieved in Java as:

```
Calendar calendar = Calendar.getInstance();
int currentSecond = calendar.get(Calendar.SECOND);
```

4) ***frequencies*** array counts are incremented as:

```
int newSecond = calendar.get(Calendar.SECOND);
if (newSecond != currentSecond)
{
    synchronized (...)
    {
        int requestsPerSecond = sumFrequenciesInTheSecond (frequencies);
        frequencies = new AtomicInteger[1000];
    }
}
// frequencies points to current second at this point
int requestMillisOfSecond = calendar.get(Calendar.MILLISECOND);
frequencies[requestMillisOfSecond]++;
```

5) So we are able to get the frequencies per second by just using 1000 AtomicIntegers.

6) To get requests per minute, all we need to do is add the seconds in a minute.
This can be done by storing just 60 seconds of data, i.e. just 60 integers.

7) Similarly, once we have minutes' data, we can get hours' data by adding 60 minutes.
Again there is no need to store more than 60 minutes of data.
Whenever 60 minutes complete for the current hour, we sum up those minutes, store the hour's requests per second and reset the minutes array.

8) Thus, all we need is the following:

```
AtomicInteger frequencies[1000];
int secondsFrequencies[60]; // Every second gets its value by summing frequencies array
int minutesFrequencies[60]; // Every minute gets its value by summing secondsFrequencies array
int hoursFrequencies[60]; // Every hour gets its value by summing minutesFrequencies array
```

9) This way, we get our solution in just O(1) space