Trabalho 07

1. Implemente um programa de contagem estatística (conforme descrito no Perfbook). Seu programa deve executar um número N de *threads* contadoras (TCs) parametrizável. Estas funcionam em um laço, incrementando seus contadores locais a cada iteração. Além disso, deve executar uma *thread* leitora responsável por ler os valores atuais dos contadores das TCs e produzir uma soma global desses valores, imprimindo-a. A execução deve parar quando o valor da soma total atingir um limite K, também parametrizável. Execute esse programa para valores de K maiores que $K \geq 2^{31}$ e meça o tempo de execução. Qual o efeito de tornar *volatile* o contador de cada *thread*? O tempo de execução muda? O tipo do atributo contador influencia esse tempo de execução? Compare os resultados para *int*, *float*, *double* e *long*. Os intervalos entre leituras dos contadores de cada *thread* influencia o tempo total da execução? Para medir o tempo, realize pelo menos dez execuções e use a média das últimas três execuções como seu tempo oficial.

Os arquivos a seguir foram criados para representar o problema especificado na questão. O limite K e o número de *threads* contadoras N são representados pelas constantes de mesmo nome em StatisticalCounter. O tempo total de execução é medido a partir da criação das *threads* até o momento em que todas anunciam sua finalização.

Detalhes da execução serão mostrados depois dos arquivos.

```
public class StatisticalCounterMain {

   public static void main(String[] args) {
      StatisticalCounter counter = new StatisticalCounter();

      // create the thread that will be responsible to read the counter
      Thread readerThread = new Thread(new StatisticalReaderRunnable(counter));
      readerThread.start();

      counter.start();
   }
}
```

```
public class StatisticalReaderRunnable implements Runnable {

   private StatisticalCounter counter;
```

```
    public StatisticalReaderRunnable(StatisticalCounter counter) {
        this.counter = counter;
    }

    @Override
    public void run() {
        long current = 0;

        do {
            current = this.counter.read();
            System.out.println("Current counter value is: " + current);
        } while (!this.counter.isLimitReached());
    }
}
```

```
public interface StatisticalCounterListener {
    public void onExecutionFinished(int counterId);
}
```

```
public class StatisticalCounterRunnable implements Runnable {

    // id to identify the counter
    private int counterId;

    // local counter
    private long counter = 0;

    // listener to be called when execution is finished
    private StatisticalCounterListener listener;

    // flag to identify if the limit was reached
    private volatile boolean limitReached = false;

    public StatisticalCounterRunnable(int counterId, StatisticalCounterListener listener) {
        this.counterId = counterId;
        this.listener = listener;
    }

    @Override
```

```
    public void run() {
        // increment the counter while the limit is not reached
        while (!this.limitReached) {
            this.counter++;
        }

        // tell someone that the execution was finished
        if (this.listener != null) {
            this.listener.onExecutionFinished(this.counterId);
        }
    }

    public long getCounterValue() {
        return this.counter;
    }

    public void setLimitReached(boolean limitReached) {
        this.limitReached = limitReached;
    }
}
```

```
package statistical;

public class StatisticalCounter {

    // number of threads that will increment the counter
    private static final int N = 100;

    // the limit to be reached by counter
    private static final long K = (long) Math.pow(2, 33);

    // flag to identify if the limit was reached on the sum of thread counters
    private volatile boolean limitReached = false;

    // the counters that will provide the total value of the current counter
    private StatisticalCounterRunnable[] counters;

    // variables to identify execution time
    private long startTime;
    private int unregisteredCounters = 0;

    public StatisticalCounter() {
        // just create the array of counters
        this.counters = new StatisticalCounterRunnable[N];
    }
```

```
public void start() {
   this.startTime = System.currentTimeMillis();

   // create the N counter threads and register it on counters list
   for (int i = 0; i < this.counters.length; i++) {
      this.counters[i] = new StatisticalCounterRunnable(i, listener);
      Thread counterThread = new Thread(this.counters[i]);
      counterThread.start();
   }
}

public synchronized long read() {
   long current = 0;

   // iterate over counters in order to retrieve the total
   for (int i = 0; i < this.counters.length; i++) {
      if (this.counters[i] != null) {
         current += this.counters[i].getCounterValue();
      }
   }

   // verify if the limit was reached
   if (current >= K || current < 0) {
      this.tellEveryoneTheLimitWasReached();
   }

   return current;
}

public boolean isLimitReached() {
   return this.limitReached;
}

private void tellEveryoneTheLimitWasReached() {
   for (int i = 0; i < this.counters.length; i++) {
      if (this.counters[i] != null) {
         this.counters[i].setLimitReached(true);
      }
   }

   this.limitReached = true;
}

private synchronized void unregisterCounter(int id, long current) {
   unregisteredCounters++;
   if (unregisteredCounters >= N) {
```

```
            float spentTime = (float)(System.currentTimeMillis() - startTime) / 1000f;
            System.out.println("Time spent: " + spentTime + " seconds");

            if (current < 0) {
                System.out.println("Dude, it happened an overflow!");
            }
        }
    }

    // it is just a listener to identify when the thread was finished
    private StatisticalCounterListener listener = new StatisticalCounterListener() {

        @Override
        public void onExecutionFinished(int counterId) {
            long current = read();
            System.out.println("Counter " + counterId + " finished and value read was " + current);
            unregisterCounter(counterId, current);
        }
    };
}
```

1.1. Execute esse programa para valores de K maiores que $K \geq 2^{31}$ e meça o tempo de execução.

Inicialmente o programa foi executado para um $K = 2^{31}$, por se tratar do limite possível de se representar com inteiro. Entretanto, aconteceu um fato interessante, um *overflow* fazia com que o valor fosse sempre menor que o limite na leitura, pois estavam sendo utilizadas variáveis do tipo *int* para acumular os contadores e representar o limite K. Ajustes seriam necessários para considerar valores maiores que 2^{31} utilizando uma variável de tipo *int*.

Para considerar um valor de 2^{33}, foi utilizada uma variável do tipo *long* para acumular os valores dos contadores de cada *thread* e o limite K. Os testes consideraram a execução de 100 *threads* contadoras.

Tempo de execução (em segundos) para dez execuções diferentes.

2,677	2,564	2,626	2,91	2,569	2,786	2,731	2,583	2,728	2,578
							Média = 2,629		

1.2. Qual o efeito de tornar *volatile* o contador de cada *thread*? O tempo de execução muda?

Mudando o contador de cada *thread* para *volatile* resultou em um aumento considerável no tempo de execução, como mostrado na tabela abaixo.

Tempo de execução (em segundos) para dez execuções diferentes.

14,39	14,311	14,331	14,206	14,358	14,191	14,295	14,189	14,322	14,258
							Média = 14,256		

1.3. O tipo do atributo contador influencia esse tempo de execução? Compare os resultados para *int*, *float*, *double* e *long*.

Para comparar o uso de *int*, *float*, *double* e *long*, ocorreram as seguintes adaptações no código:

- A classe StatisticalCounterRunnable sofreu alteração no tipo da variável counter.
- O read() em StatisticalCounter foi trocado para *int*.
- O número de *threads* contadoras N foi mantido como 100.
- O valor de K foi modificado para um inteiro de valor 1.600.000.000, que seria algo próximo do maior valor inteiro representado pelas 100 *threads* utilizando *float* (1.677.721.600), algumas *threads* podem parar de incrementar antes do final da execução no caso desse tipo de variável.

Tempo de execução (em segundos) para dez execuções diferentes utilizando *int*.

0,663	0,584	0,651	0,54	0,5	0,579	0,619	0,614	0,561	0,549
							Média = 0,574		

Tempo de execução (em segundos) para dez execuções diferentes utilizando *float*.

0,723	0,772	0,829	0,79	0,68	0,684	0,709	0,777	0,801	0,749
							Média = 0,775		

Tempo de execução (em segundos) para dez execuções diferentes utilizando *double*.

0,752	0,65	0,691	0,813	0,666	0,728	0,885	0,726	0,697	0,743
							Média = 0,722		

Tempo de execução (em segundos) para dez execuções diferentes utilizando *long*.

0,585	0,726	0,528	0,504	0,573	0,656	0,688	0,642	0,61	0,593
							Média = 0,615		

Comparando as médias de tempo de execução é possível notar que variáveis de tipo *float* e *double* apresentaram uma performance inferior.

1.4. Os intervalos entre leituras dos contadores de cada *thread* influencia o tempo total da execução?

O código foi alterado para chamada de um *Thread.sleep(20)* em cada execução de leitura. Com isso, a cada leitura será esperado um tempo de 20 milisegundos para continuar a execução. O resultado em tempo de execução foi ligeiramente aprimorado em relação à média apresentada anteriormente de 2,629 segundos, conforme visto na tabela abaixo.

Tempo de execução (em segundos) para dez execuções diferentes utilizando *AtomicInteger*.

2,516	2,506	2,549	2,532	2,51	2,615	2,586	2,541	2,597	2,52
							Média = 2,552		

2. Verifique, para o programa do exemplo anterior, o efeito de usar contadores dos tipos *AtomicInteger* e/ou *AtomicLong* no tempo de execução.

Para a execução dos testes abaixo a variável *counter* em StatisticalCounterRunnable foram substituídas pelas classes correspondentes. O valor foi incrementado através da função *incrementAndGet* e obtido através da função *get*.

Tempo de execução (em segundos) para dez execuções diferentes utilizando *AtomicInteger*.

17,481	17,852	17,679	17,561	17,912	17,565	17,967	17,997	17,721	18,07
							Média = 17,929		

Tempo de execução (em segundos) para dez execuções diferentes utilizando *AtomicLong*.

18,177	18,041	17,724	18,1	17,774	18,162	17,569	17,856	17,694	17,784
							Média = 17,778		

É possível notar que a execução de contadores que utilizam *AtomicInteger* e *AtomicLong* apresentaram uma performance bastante inferior à execução utilizando a variável não atômica de tipo *long*, que foi apresentada em uma média de 2,629 segundos.

3. Torne exato (ou seja, não mais estatístico) o contador do item 1. Torná-lo exato significa que a soma total não pode passar de K. Uma abordagem de fastpath pode ajudar neste caso? Em caso afirmativo, mostre como. Em caso negativo, mostre porque não. Qual o desempenho desse contador? Empregue a mesma metodologia de medição descrita nos itens anteriores.

[...]