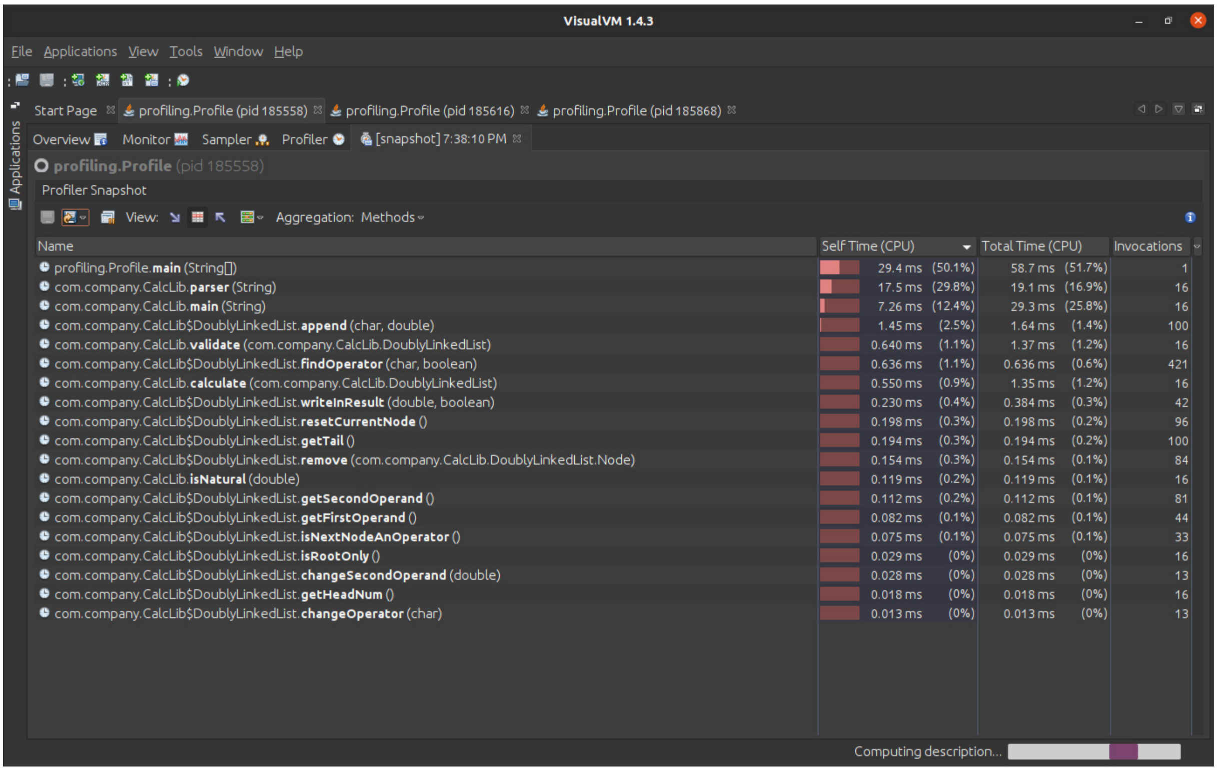
Profiling the calculator

Results report

Patrik Skaloš, xskalo01



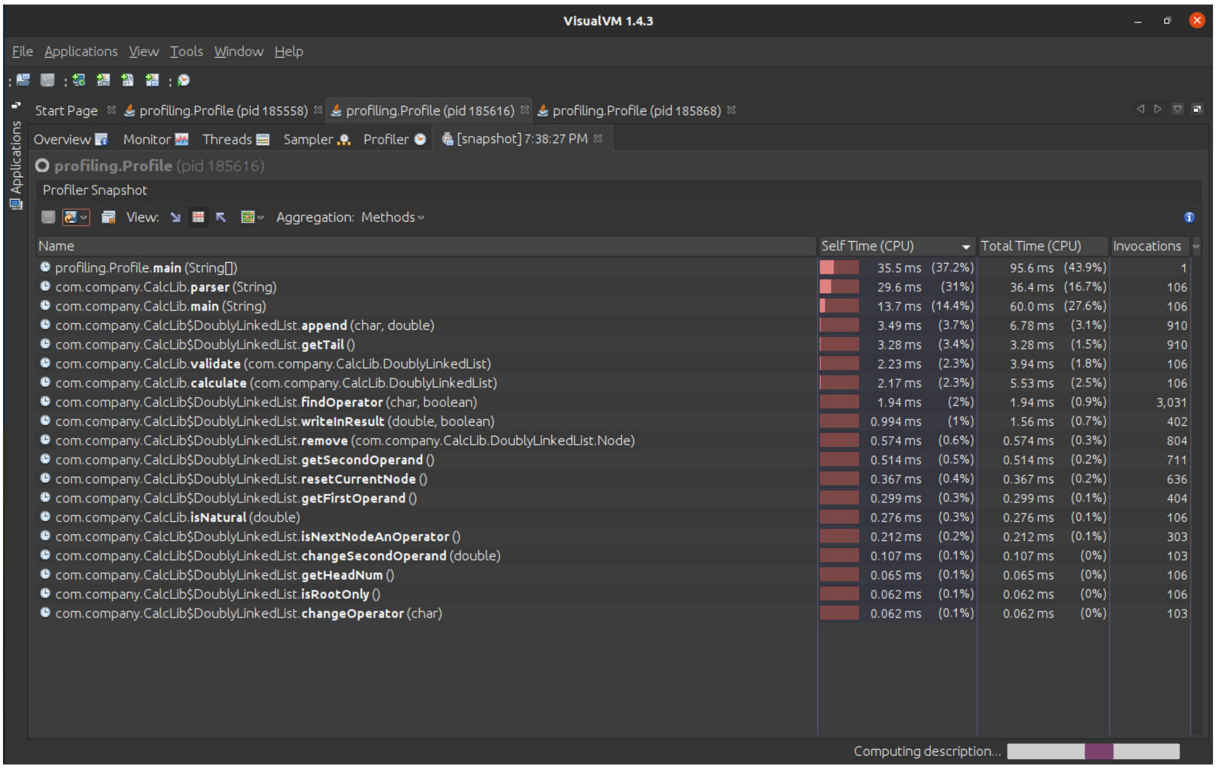
VisualVM 1.4.3
File Applications View Tools Window Help
Start Page
profiling.Profile (pid 185558)
profiling.Profile (pid 185616)
profiling.Profile (pid 185868)
Applications
Overview
Monitor
Sampler
Profiler
[snapshot] 7:38:10 PM
profiling.Profile (pid 185558)
Profiler Snapshot
View:
Aggregation: Methods
Computing description...

Name	Self Time (CPU)	Total Time (CPU)	Invocations
profiling.Profile.**main** (String[])	29.4 ms (50.1%)	58.7 ms (51.7%)	1
com.company.CalcLib.**parser** (String)	17.5 ms (29.8%)	19.1 ms (16.9%)	16
com.company.CalcLib.**main** (String)	7.26 ms (12.4%)	29.3 ms (25.8%)	16
com.company.CalcLib$DoublyLinkedList.**append** (char, double)	1.45 ms (2.5%)	1.64 ms (1.4%)	100
com.company.CalcLib.**validate** (com.company.CalcLib.DoublyLinkedList)	0.640 ms (1.1%)	1.37 ms (1.2%)	16
com.company.CalcLib$DoublyLinkedList.**findOperator** (char, boolean)	0.636 ms (1.1%)	0.636 ms (0.6%)	421
com.company.CalcLib.**calculate** (com.company.CalcLib.DoublyLinkedList)	0.550 ms (0.9%)	1.35 ms (1.2%)	16
com.company.CalcLib$DoublyLinkedList.**writeInResult** (double, boolean)	0.230 ms (0.4%)	0.384 ms (0.3%)	42
com.company.CalcLib$DoublyLinkedList.**resetCurrentNode** ()	0.198 ms (0.3%)	0.198 ms (0.2%)	96
com.company.CalcLib$DoublyLinkedList.**getTail** ()	0.194 ms (0.3%)	0.194 ms (0.2%)	100
com.company.CalcLib$DoublyLinkedList.**remove** (com.company.CalcLib.DoublyLinkedList.Node)	0.154 ms (0.3%)	0.154 ms (0.1%)	84
com.company.CalcLib.**isNatural** (double)	0.119 ms (0.2%)	0.119 ms (0.1%)	16
com.company.CalcLib$DoublyLinkedList.**getSecondOperand** ()	0.112 ms (0.2%)	0.112 ms (0.1%)	81
com.company.CalcLib$DoublyLinkedList.**getFirstOperand** ()	0.082 ms (0.1%)	0.082 ms (0.1%)	44
com.company.CalcLib$DoublyLinkedList.**isNextNodeAnOperator** ()	0.075 ms (0.1%)	0.075 ms (0.1%)	33
com.company.CalcLib$DoublyLinkedList.**isRootOnly** ()	0.029 ms (0%)	0.029 ms (0%)	16
com.company.CalcLib$DoublyLinkedList.**changeSecondOperand** (double)	0.028 ms (0%)	0.028 ms (0%)	13
com.company.CalcLib$DoublyLinkedList.**getHeadNum** ()	0.018 ms (0%)	0.018 ms (0%)	16
com.company.CalcLib$DoublyLinkedList.**changeOperator** (char)	0.013 ms (0%)	0.013 ms (0%)	13

Figure 1: Output of the profiler when calculating the standard deviation of **10** numbers

In the picture we can clearly see that the library spent the most time parsing the input (we should ignore the `profiling.Profile.main` method). This is probably because of time-consuming operations with strings (mainly when a root or a power is to be calculated). It still only took 17 milliseconds, so I will conclude this after watching subsequent outputs.

The `main` method also took some time (7 milliseconds), which is surprising, because it is *fairly empty*. I suppose the only thing that could take a long time is figuring out in which format to print the result in. For example, instead of comparing the result to a constant, it is compared to 2^{52} (accurate range of the double data type) which has to be computed. We will see if it changes in subsequent outputs.

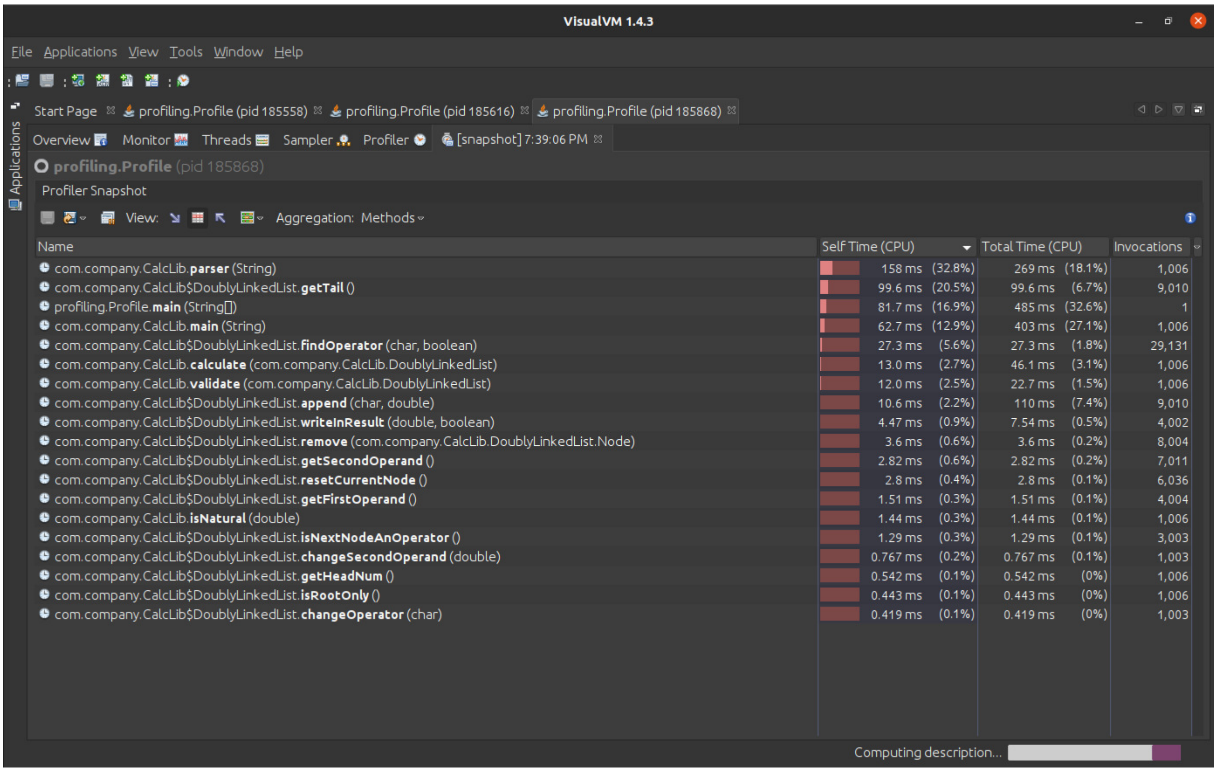

VisualVM 1.4.3
File Applications View Tools Window Help
Start Page
profiling.Profile (pid 185558)
profiling.Profile (pid 185616)
profiling.Profile (pid 185868)
Applications
Overview Monitor Threads Sampler Profiler
[snapshot] 7:38:27 PM
profiling.Profile (pid 185616)
Profiler Snapshot
View:
Aggregation: Methods
Computing description...

Name	Self Time (CPU)		Total Time (CPU)		Invocations
profiling.Profile.**main** (String[])	35.5 ms	(37.2%)	95.6 ms	(43.9%)	1
com.company.CalcLib.**parser** (String)	29.6 ms	(31%)	36.4 ms	(16.7%)	106
com.company.CalcLib.**main** (String)	13.7 ms	(14.4%)	60.0 ms	(27.6%)	106
com.company.CalcLib$DoublyLinkedList.**append** (char, double)	3.49 ms	(3.7%)	6.78 ms	(3.1%)	910
com.company.CalcLib$DoublyLinkedList.**getTail** ()	3.28 ms	(3.4%)	3.28 ms	(1.5%)	910
com.company.CalcLib.**validate** (com.company.CalcLib.DoublyLinkedList)	2.23 ms	(2.3%)	3.94 ms	(1.8%)	106
com.company.CalcLib.**calculate** (com.company.CalcLib.DoublyLinkedList)	2.17 ms	(2.3%)	5.53 ms	(2.5%)	106
com.company.CalcLib$DoublyLinkedList.**findOperator** (char, boolean)	1.94 ms	(2%)	1.94 ms	(0.9%)	3,031
com.company.CalcLib$DoublyLinkedList.**writeInResult** (double, boolean)	0.994 ms	(1%)	1.56 ms	(0.7%)	402
com.company.CalcLib$DoublyLinkedList.**remove** (com.company.CalcLib.DoublyLinkedList.Node)	0.574 ms	(0.6%)	0.574 ms	(0.3%)	804
com.company.CalcLib$DoublyLinkedList.**getSecondOperand** ()	0.514 ms	(0.5%)	0.514 ms	(0.2%)	711
com.company.CalcLib$DoublyLinkedList.**resetCurrentNode** ()	0.367 ms	(0.4%)	0.367 ms	(0.2%)	636
com.company.CalcLib$DoublyLinkedList.**getFirstOperand** ()	0.299 ms	(0.3%)	0.299 ms	(0.1%)	404
com.company.CalcLib.**isNatural** (double)	0.276 ms	(0.3%)	0.276 ms	(0.1%)	106
com.company.CalcLib$DoublyLinkedList.**isNextNodeAnOperator** ()	0.212 ms	(0.2%)	0.212 ms	(0.1%)	303
com.company.CalcLib$DoublyLinkedList.**changeSecondOperand** (double)	0.107 ms	(0.1%)	0.107 ms	(0%)	103
com.company.CalcLib$DoublyLinkedList.**getHeadNum** ()	0.065 ms	(0.1%)	0.065 ms	(0%)	106
com.company.CalcLib$DoublyLinkedList.**isRootOnly** ()	0.062 ms	(0.1%)	0.062 ms	(0%)	106
com.company.CalcLib$DoublyLinkedList.**changeOperator** (char)	0.062 ms	(0.1%)	0.062 ms	(0%)	103

Figure 2: Output of the profiler when calculating the standard deviation of **100** numbers

We still see the `parser` method at the top (ignoring the `profiling.Profile.main` method), which means that it is definitely worth optimising. Optimisation should be probably focused on more efficient ways of working with strings, as there are actually too many calls (mainly) for the `replace` method in the code.

Six times more calls to the `main` method of `CalcLib` class were made compared to the previous figure (which was run with 10 times less numbers), and the time doubled. This does not sound so bad, even when scaled, but it is definitely worth looking into. I suppose it is taking too long exactly because of the computation of 2^{52}, as I mentioned in the last paragraph. In this figure the `main` method from `CalcLib` class was called 106 times, which means that 2^{52} was calculated 106 times...


VisualVM 1.4.3
File Applications View Tools Window Help
Start Page
profiling.Profile (pid 185558)
profiling.Profile (pid 185616)
profiling.Profile (pid 185868)
Applications
Overview Monitor Threads Sampler Profiler
[snapshot] 7:39:06 PM
profiling.Profile (pid 185868)
Profiler Snapshot
View:
Aggregation: Methods
Computing description...

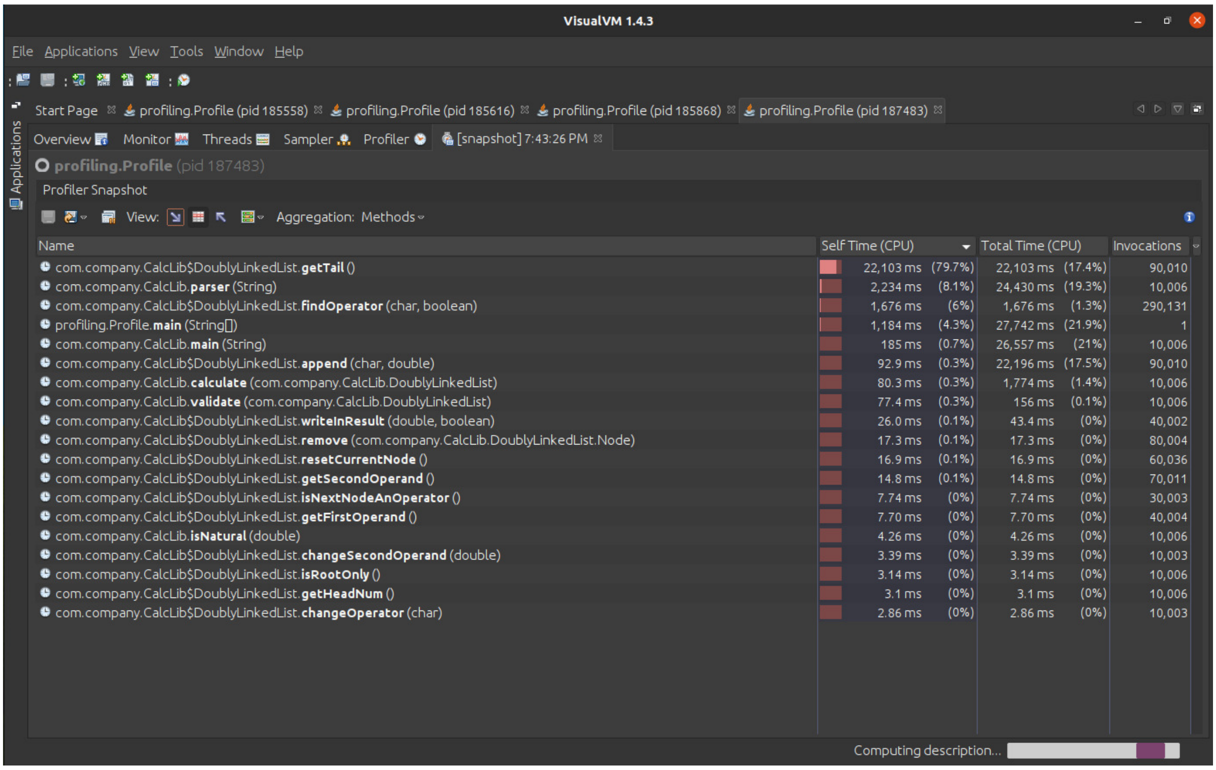
Name	Self Time (CPU)		Total Time (CPU)		Invocations
com.company.CalcLib.**parser** (String)	158 ms	(32.8%)	269 ms	(18.1%)	1,006
com.company.CalcLib$DoublyLinkedList.**getTail** ()	99.6 ms	(20.5%)	99.6 ms	(6.7%)	9,010
profiling.Profile.**main** (String[])	81.7 ms	(16.9%)	485 ms	(32.6%)	1
com.company.CalcLib.**main** (String)	62.7 ms	(12.9%)	403 ms	(27.1%)	1,006
com.company.CalcLib$DoublyLinkedList.**findOperator** (char, boolean)	27.3 ms	(5.6%)	27.3 ms	(1.8%)	29,131
com.company.CalcLib.**calculate** (com.company.CalcLib.DoublyLinkedList)	13.0 ms	(2.7%)	46.1 ms	(3.1%)	1,006
com.company.CalcLib.**validate** (com.company.CalcLib.DoublyLinkedList)	12.0 ms	(2.5%)	22.7 ms	(1.5%)	1,006
com.company.CalcLib$DoublyLinkedList.**append** (char, double)	10.6 ms	(2.2%)	110 ms	(7.4%)	9,010
com.company.CalcLib$DoublyLinkedList.**writeInResult** (double, boolean)	4.47 ms	(0.9%)	7.54 ms	(0.5%)	4,002
com.company.CalcLib$DoublyLinkedList.**remove** (com.company.CalcLib.DoublyLinkedList.Node)	3.6 ms	(0.6%)	3.6 ms	(0.2%)	8,004
com.company.CalcLib$DoublyLinkedList.**getSecondOperand** ()	2.82 ms	(0.6%)	2.82 ms	(0.2%)	7,011
com.company.CalcLib$DoublyLinkedList.**resetCurrentNode** ()	2.8 ms	(0.4%)	2.8 ms	(0.1%)	6,036
com.company.CalcLib$DoublyLinkedList.**getFirstOperand** ()	1.51 ms	(0.3%)	1.51 ms	(0.1%)	4,004
com.company.CalcLib.**isNatural** (double)	1.44 ms	(0.3%)	1.44 ms	(0.1%)	1,006
com.company.CalcLib$DoublyLinkedList.**isNextNodeAnOperator** ()	1.29 ms	(0.3%)	1.29 ms	(0.1%)	3,003
com.company.CalcLib$DoublyLinkedList.**changeSecondOperand** (double)	0.767 ms	(0.2%)	0.767 ms	(0.1%)	1,003
com.company.CalcLib$DoublyLinkedList.**getHeadNum** ()	0.542 ms	(0.1%)	0.542 ms	(0%)	1,006
com.company.CalcLib$DoublyLinkedList.**isRootOnly** ()	0.443 ms	(0.1%)	0.443 ms	(0%)	1,006
com.company.CalcLib$DoublyLinkedList.**changeOperator** (char)	0.419 ms	(0.1%)	0.419 ms	(0%)	1,003

Figure 3: Output of the profiler when calculating the standard deviation of **1 000** numbers

Calculating the standard deviation of 1000 numbers (mind that the numbers are within range <0, 1 000 000>) is starting to take some time.

The `parser` method still occupies the top of the list, which only proves my point it needs to be optimised.

Surprisingly, though, `getTail` method jumped high and took 100 milliseconds, which, to me, sounds like too long. I highly doubt a human would enter such a long expression for this to be a problem, but optimising this part should be really easy (although, the library could be used as a part of a different software, in which case, this definitely is a problem). The simplest and very efficient way to fix this would be to have a `Node tail` variable, which would be updated before every appending or removal of a node. That way, `getTail` method would only return the `Node` pointer instead of searching through the whole list to find the end...


VisualVM 1.4.3
File Applications View Tools Window Help
Start Page
profiling.Profile (pid 185558)
profiling.Profile (pid 185616)
profiling.Profile (pid 185868)
profiling.Profile (pid 187483)
Applications
Overview
Monitor
Threads
Sampler
Profiler
[snapshot] 7:43:26 PM
profiling.Profile (pid 187483)
Profiler Snapshot
View:
Aggregation: Methods
Computing description...

Name	Self Time (CPU)		Total Time (CPU)		Invocations
com.company.CalcLib$DoublyLinkedList.**getTail** ()	22,103 ms	(79.7%)	22,103 ms	(17.4%)	90,010
com.company.CalcLib.**parser** (String)	2,234 ms	(8.1%)	24,430 ms	(19.3%)	10,006
com.company.CalcLib$DoublyLinkedList.**findOperator** (char, boolean)	1,676 ms	(6%)	1,676 ms	(1.3%)	290,131
profiling.Profile.**main** (String[])	1,184 ms	(4.3%)	27,742 ms	(21.9%)	1
com.company.CalcLib.**main** (String)	185 ms	(0.7%)	26,557 ms	(21%)	10,006
com.company.CalcLib$DoublyLinkedList.**append** (char, double)	92.9 ms	(0.3%)	22,196 ms	(17.5%)	90,010
com.company.CalcLib.**calculate** (com.company.CalcLib.DoublyLinkedList)	80.3 ms	(0.3%)	1,774 ms	(1.4%)	10,006
com.company.CalcLib.**validate** (com.company.CalcLib.DoublyLinkedList)	77.4 ms	(0.3%)	156 ms	(0.1%)	10,006
com.company.CalcLib$DoublyLinkedList.**writeInResult** (double, boolean)	26.0 ms	(0.1%)	43.4 ms	(0%)	40,002
com.company.CalcLib$DoublyLinkedList.**remove** (com.company.CalcLib.DoublyLinkedList.Node)	17.3 ms	(0.1%)	17.3 ms	(0%)	80,004
com.company.CalcLib$DoublyLinkedList.**resetCurrentNode** ()	16.9 ms	(0.1%)	16.9 ms	(0%)	60,036
com.company.CalcLib$DoublyLinkedList.**getSecondOperand** ()	14.8 ms	(0.1%)	14.8 ms	(0%)	70,011
com.company.CalcLib$DoublyLinkedList.**isNextNodeAnOperator** ()	7.74 ms	(0%)	7.74 ms	(0%)	30,003
com.company.CalcLib$DoublyLinkedList.**getFirstOperand** ()	7.70 ms	(0%)	7.70 ms	(0%)	40,004
com.company.CalcLib.**isNatural** (double)	4.26 ms	(0%)	4.26 ms	(0%)	10,006
com.company.CalcLib$DoublyLinkedList.**changeSecondOperand** (double)	3.39 ms	(0%)	3.39 ms	(0%)	10,003
com.company.CalcLib$DoublyLinkedList.**isRootOnly** ()	3.14 ms	(0%)	3.14 ms	(0%)	10,006
com.company.CalcLib$DoublyLinkedList.**getHeadNum** ()	3.1 ms	(0%)	3.1 ms	(0%)	10,006
com.company.CalcLib$DoublyLinkedList.**changeOperator** (char)	2.86 ms	(0%)	2.86 ms	(0%)	10,003

Figure 4: Output of the profiler when calculating the standard deviation of **10 000** numbers

I understand that this is not a part of my job, but I wanted to include the profiler output for 10 000 numbers as well.

We can see that the `getTail` method is a huge problem since it took more than 22 seconds.

The `parser` method still takes a lot of time, which proves it needs optimising.

What is new here is the `findOperator` method, which took 1 676 milliseconds. It's not a big surprise, since it was called more than 290 000 times. However, I don't see a simple way to optimise this method. Luckily, this doesn't seem like a big problem. We'd have to see the profiling output after optimisation of `getTail` and `parser` to decide how severe this is.

The `main` method in the library took *only* 185 milliseconds this time, which, compared to the methods I mentioned above, is not much. I would try to optimise it anyways, but when thinking about scaling the library and while looking at these outputs, 185 milliseconds does not seem like a serious hold up.

Conclusion

Methods of the `CalcLib` class to optimise sorted by importance:

- `getTail`
- `parser`
- `main`
- `findOperator`