Anton Medvedev

Dr. Gallagher

Software Testing 2

02/22/17

Assignment 1C: Profiling

- **Problem Statement**

Profiling a java program that performs relation testing of the data piped from an andrew.cs.fit.edu server.

- **Software Construction**

In order to solve this problem, multiple solutions were developed. Tools researched included Jvm profilers such as VisualVm and bash script profiling. In the end, it was decided to use a combination of bash and system calls.

- **Analysis tools**

Proposed tool was constructed from modifying original Java Pipe program by using System Calls. Specifically *nanoTime*() and *getRuntime*() methods. Method descriptions from Oracle website are:

getRuntume() => Returns the runtime object assosicated with the current Java application. Most of the methods of the class Runtime are instance methods and must be invoked with respect to the current runtime object.

nanoTime() => Returns the current value of the running JVM's high resolution time source, in nanoseconds.

For the CPU profiling, *OperatingSystemMXBean* public interface was used. Method description from Oracle website:

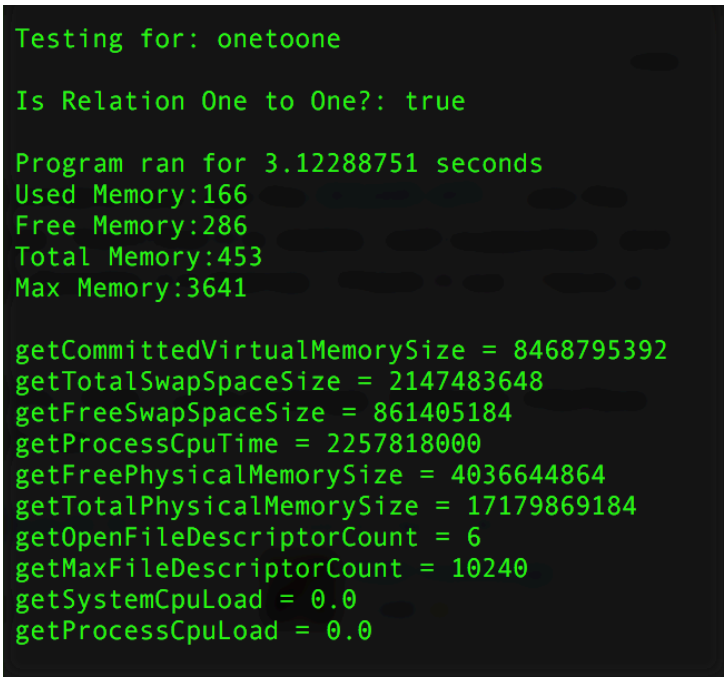
OperatingSystemMXBean extends PlatformManagedObject
=> The management interface for the operating system on which the Java virtual machine is running. A Java virtual machine has a single instance of the implementation class of this interface. This instance implementing this interface is an MXBean that can be obtained by calling the ManagementFactory.getOperatingSystemMXBean() method or from the platform MBeanServer method.

Finally bash scrip was written to perform pipe data from the server and perform analysis with different tests and arguments.

- **Program Runs**

Each program was tested with [10,5] <= [y,z] <= [1000000,50000] arguments going up by *10 each run.

Sample output format screenshot:



Sample output console results are:

Y=10 Z=5

See attached text file in “Console outputs” folder Args10/5

Y=100 Z=50

See attached text file in “Console outputs” folder Args100/50

Y=1000 Z=500

See attached text file in “Console outputs” folder Args1000/500

Y=10000 Z=5000

See attached text file in “Console outputs” folder Args10000/5000

Y=100000 Z=50000

See attached text file in “Console outputs” folder Args100000/50000

- **Statistical Analysis**

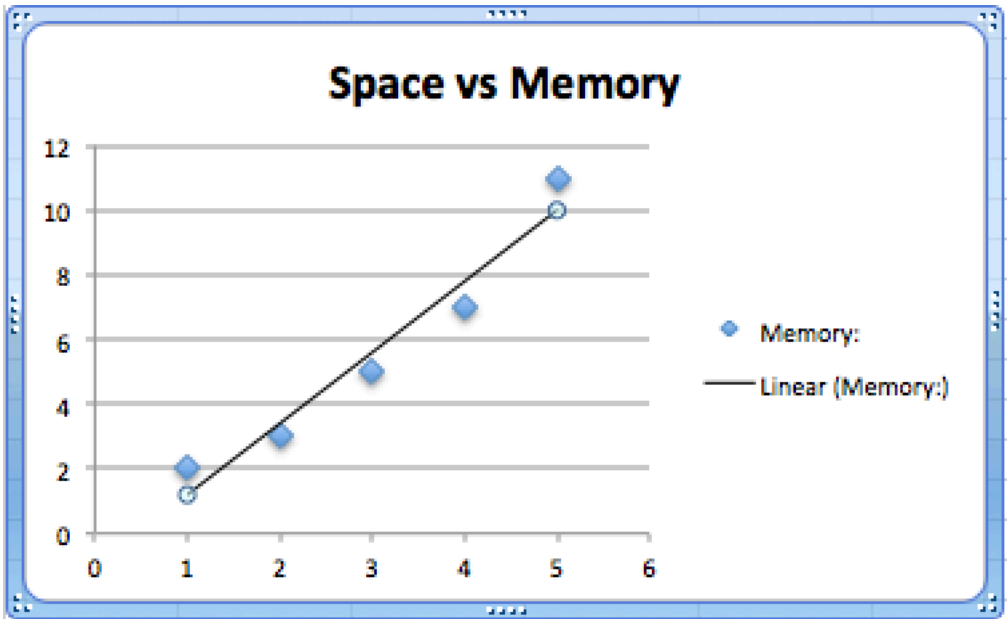
Performance predictions:

Onetoone	=> linear
Onto	=> quad
Reflex	=> linear
Sym	=> quad
Trans	=> cubic
Func	=> linear
Ref.Trans	=> cubic
Ref.Sym	=> quad
Sym.Trans	=> cubic
Eq	=> cubic to test, linear to display partitions

- **Graphs**

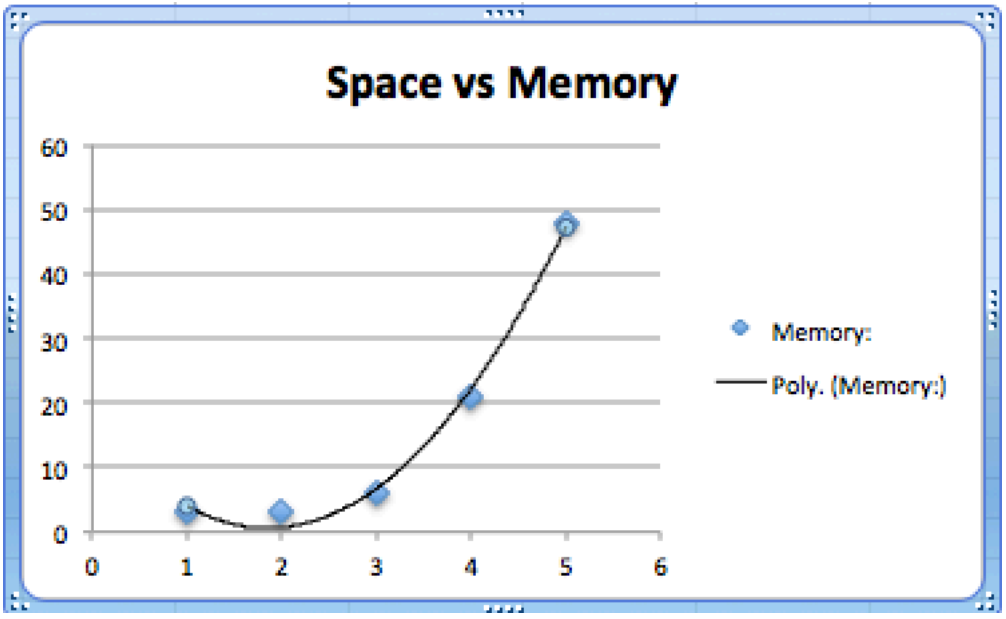
Onetoone

Linear performance as expected.



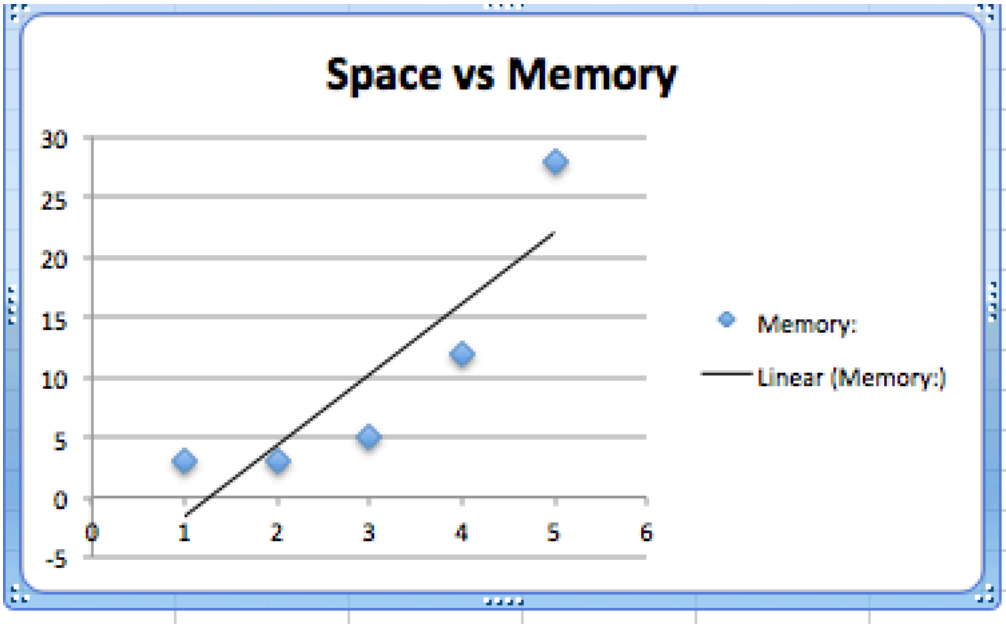
Onto

Quadratic performance as expected.



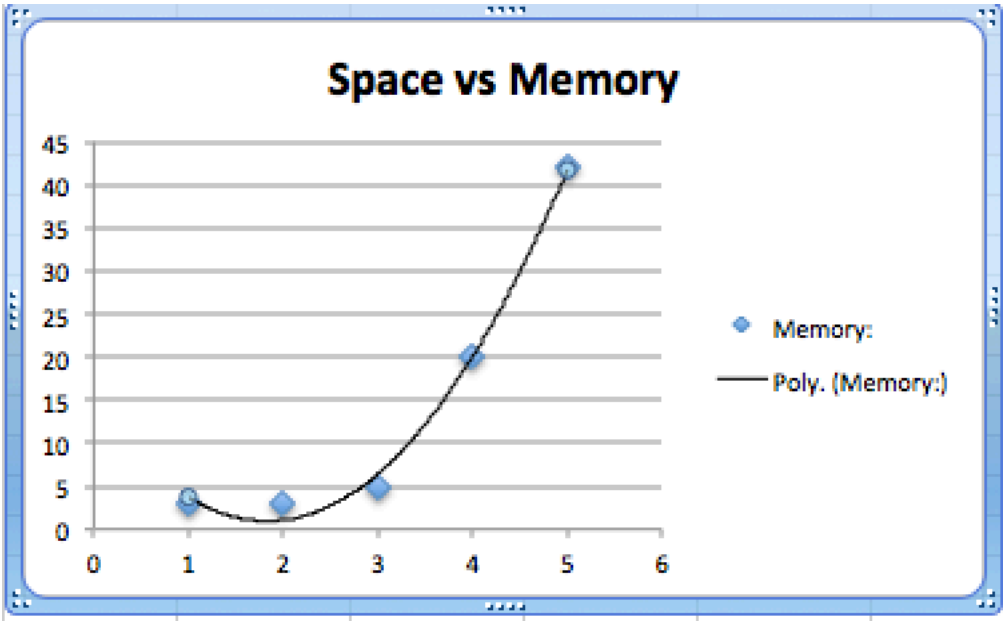
Reflex

Linear performance, though due to lower arguments does look slightly like quadratic. Anomaly should go away as *n* increases.



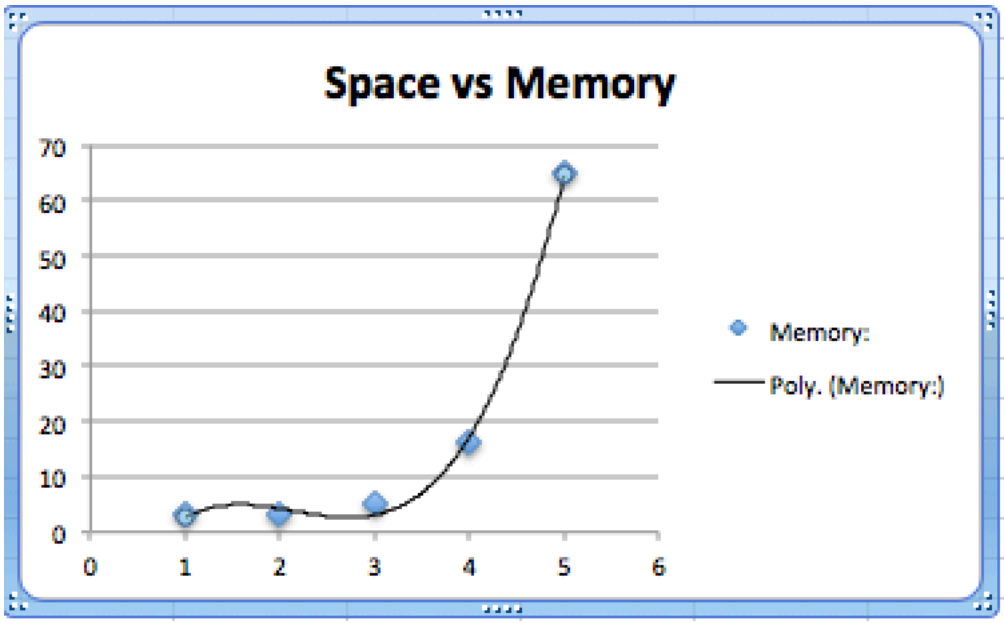
Sym

Quadratic performance as expected.



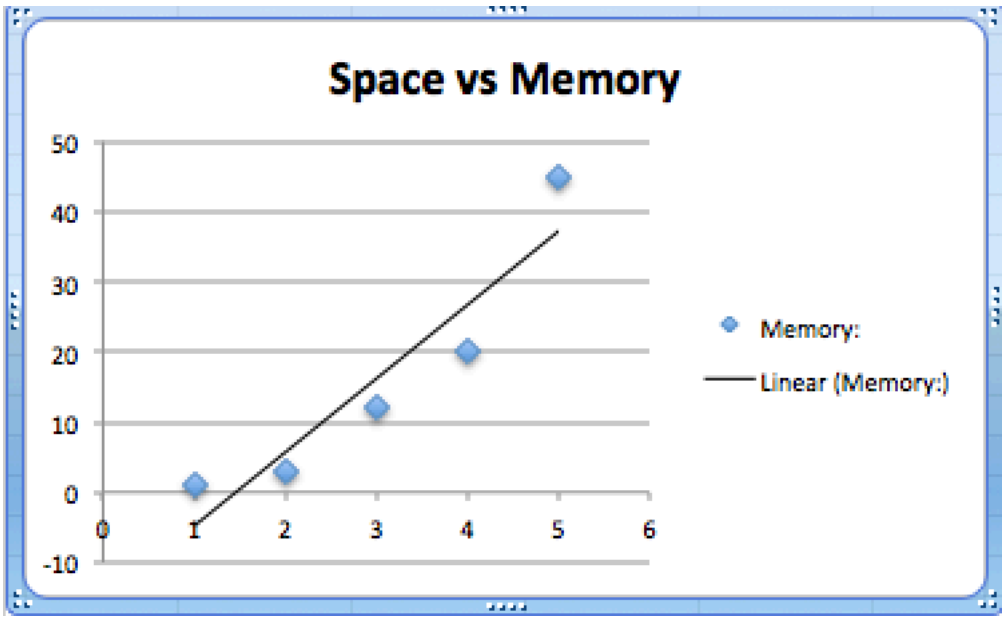
Trans

Cubic performance as expected.



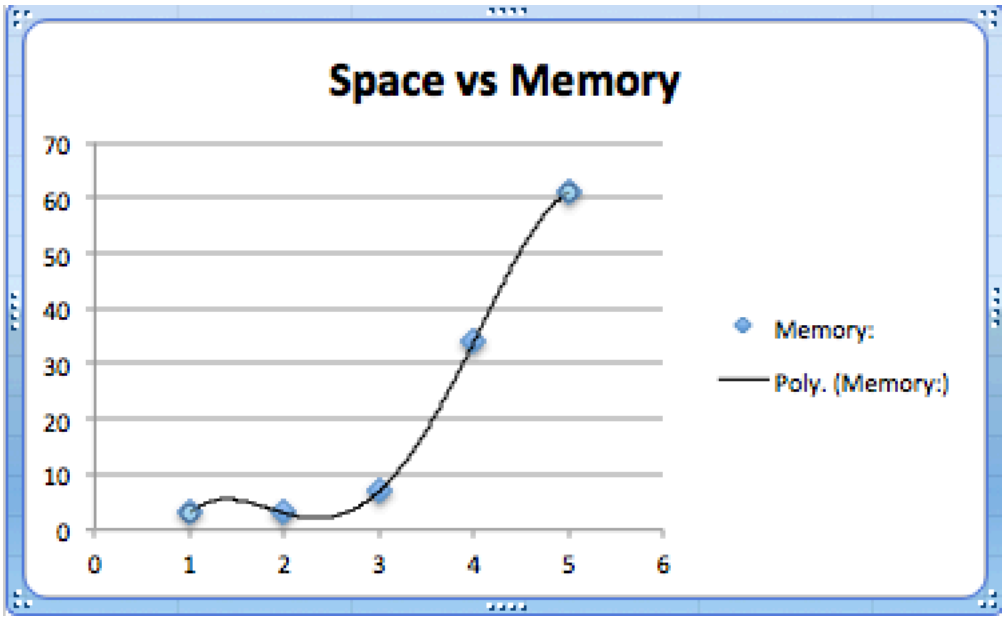
Func

Linear performance as expected.



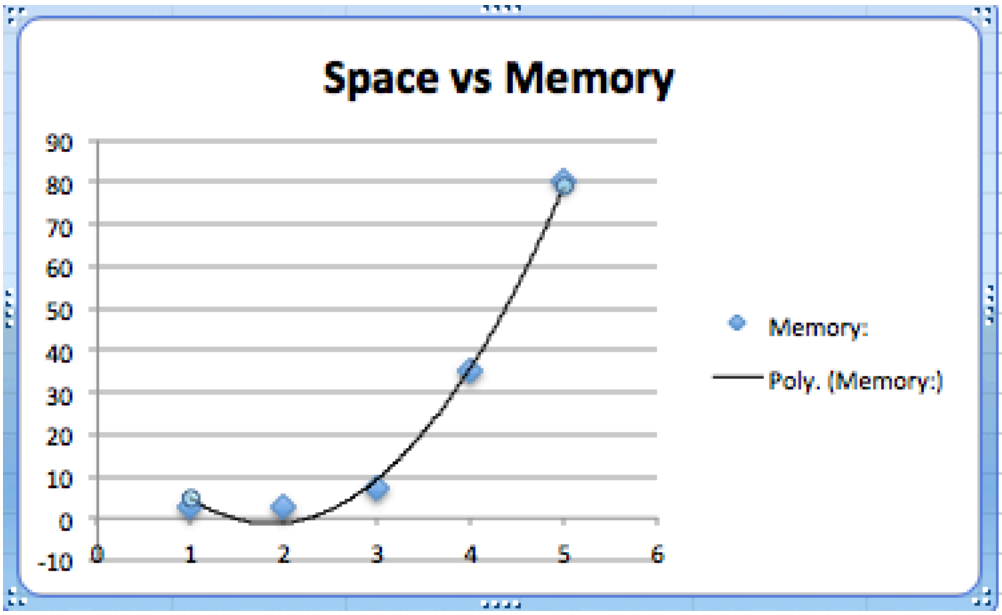
Ref.Trans

Cubic performance as expected due to transitivity check.



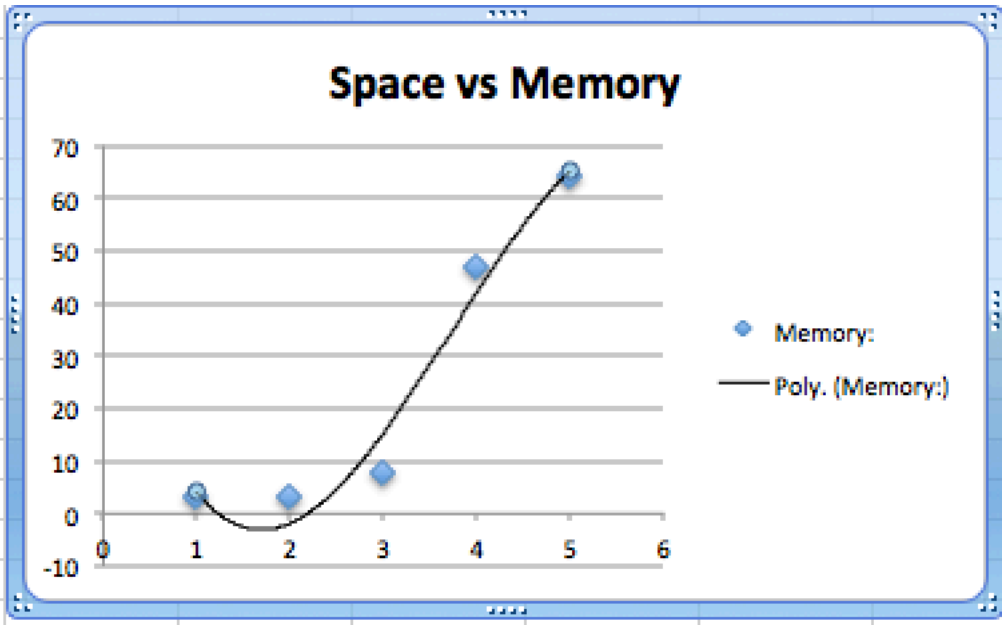
Ref.Sym

Quadratic performance as expected due to symmetry check.



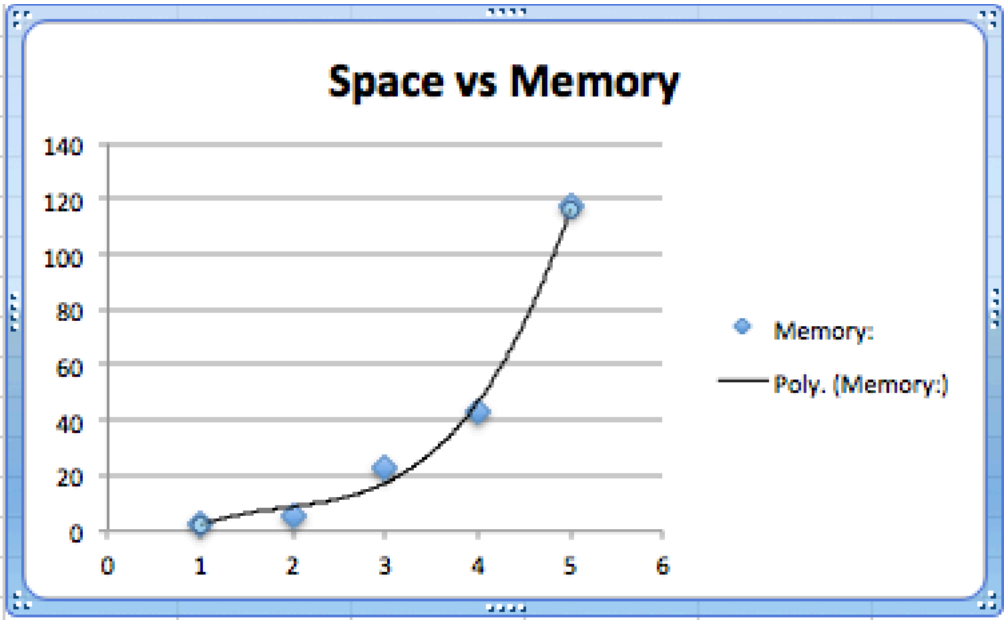
Sym.Trans

Cubic performance as expected due to transitivity check.



Eq

Cubic performance as expected due to transitivity check. Linear performance to display partitions.



References:

1)https://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#freeMemory—

2)https://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#freeMemory--

3) http://knight76.blogspot.co.il/2009/05/how-to-get-java-cpu-usage-jvm-instance.html

4) http://stackoverflow.com/questions/47177/how-to-monitor-the-computers-cpu-memory-and-disk-usage-in-java