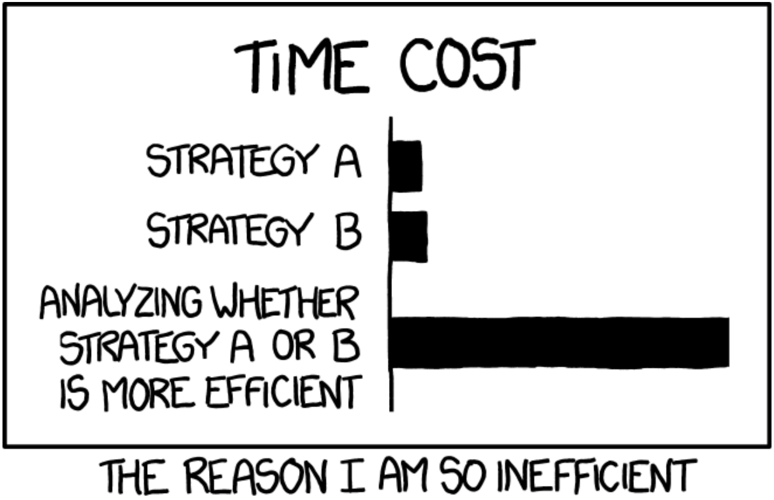

TIME COST
STRATEGY A
STRATEGY B
ANALYZING WHETHER STRATEGY A OR B IS MORE EFFICIENT
THE REASON I AM SO INEFFICIENT

Meme Credit: Randall Munroe, xkcd

Lab 4: Profiling

CS 2112 Fall 2024

September 23 / 25, 2024

Portions of today's lab slides are
adapted from CS 4152 by Prof. Walker White

Slow Operations

Many normal operations are actually relatively slow. For example:

- Instantiating Objects
- Calling Methods
- Loops

You'll learn more about why in CS 3410 and CS 4410/4

Optimization?

A common mistake is to attempt to optimize your code by not doing these slow things. One might try to make everything public, inline methods, unroll loops, etc. This is, however, a **very bad idea**.

Premature Optimization

"Premature optimization is the root of all evil"
- Donald Knuth

- Compiler automatically optimizes code
- Almost always better than what a human can do

Compiler Optimizations

Raw Code

```
int x = 8 * y;
```

Sample Compiler Output

```
int x = y << 3;
```

Compiler Optimizations

Raw Code

```java
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

Sample Compiler Output

```java
System.out.println(0);
System.out.println(1);
System.out.println(2);
System.out.println(3);
System.out.println(4);
```

Compiler Optimizations

Raw Code

```
int count = 0;
for (int i = 0; i < x; i++) {
    count++;
    doSomething();
}
```

Sample Compiler Output

```
int count = 0;
if (x > 0) {
    count = x;
    doSomething();
    for (int i = 1; i < x; i++) {
        doSomething();
    }
}
```

Credit: MSDN Magazine, 2/2015, "What Every Programmer Should Know About Compiler Optimizations"

Compiler Optimizations

Raw Code

```
int sumTo(int n) {
    int o = 0;
    for (int i = 1; i <= n; i++) {
        o += i;
    }
    return o;
}

return sumTo(10);
```

Sample Compiler Output

```
return 55;
```

Credit: Matt Godbolt, CppCon 2017, "What Has My Compiler Done for Me Lately? Unbolting the Compiler's Lid"

Compiler Optimizations

Raw Code

```
int sumTo(int n) {
    int o = 0;
    for (int i = 1; i <= n; i++) {
        o += i;
    }
    return o;
}

return sumTo(x);
```

Sample Compiler Output

```
return x + x * (x - 1) / 2;
```

Credit: Matt Godbolt, CppCon 2017, “What Has My Compiler Done for Me Lately? Unbolting the Compiler’s Lid”

Takeaway

Compilers are very smart, and do a better job making small optimizations than people generally do.
Take CS 4120 Compilers with Professor Myers to learn more.

Tuning Performance

- Don't overtune some inputs at the expense of others
- Be very cautious of making non-modular changes
- **Focus on overall algorithm first**

80/20 Rule

$\sim$ 80% of the time is spent in $\sim$ 20% of the code

The Real Question: What's the 20%?

Profiler

A profiler is a tool used to measure the performance of code

What Can We Measure?

Time

- What code takes longest
- What's called most often
- Who's calling what

Memory

- Number of objects in memory
- Size of objects in memory

How to Measure Code

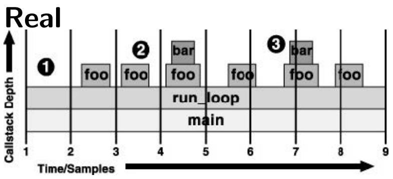
Sampling

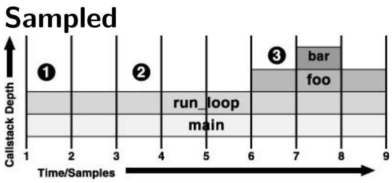
- Sample at periodic intervals
- Low overhead
- May miss small things

Instrumentation

- Count at specified places
- Gives exact view of specified slice
- Targeted

Time-Sampling





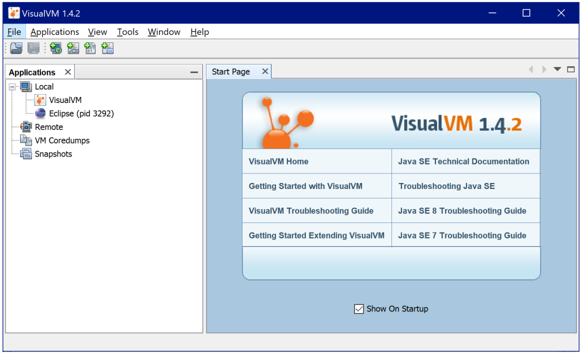
Modern profilers fix with random sampling

VisualVM

VisualVM is a Java profiler

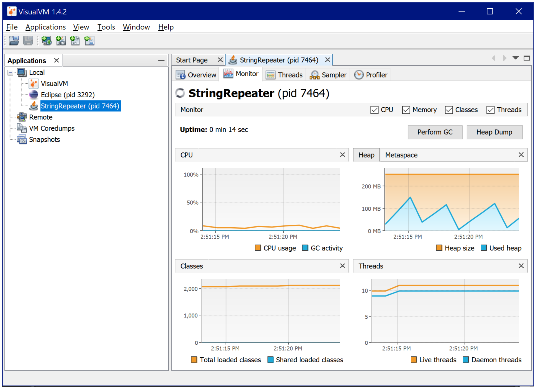
Get started by downloading it here: https://visualvm.github.io/.

VisualVM Interface


VisualVM 1.4.2
File Applications View Tools Window Help
Applications
Local
VisualVM
Eclipse (pid 3292)
Remote
VM Coredumps
Snapshots
Start Page
VisualVM 1.4.2
VisualVM Home
Java SE Technical Documentation
Getting Started with VisualVM
Troubleshooting Java SE
VisualVM Troubleshooting Guide
Java SE 8 Troubleshooting Guide
Getting Started Extending VisualVM
Java SE 7 Troubleshooting Guide
Show On Startup

VisualVM will automatically detect all running Java processes on your computer, with no additional setup required. They will be listed on the left, under Applications.

Monitor

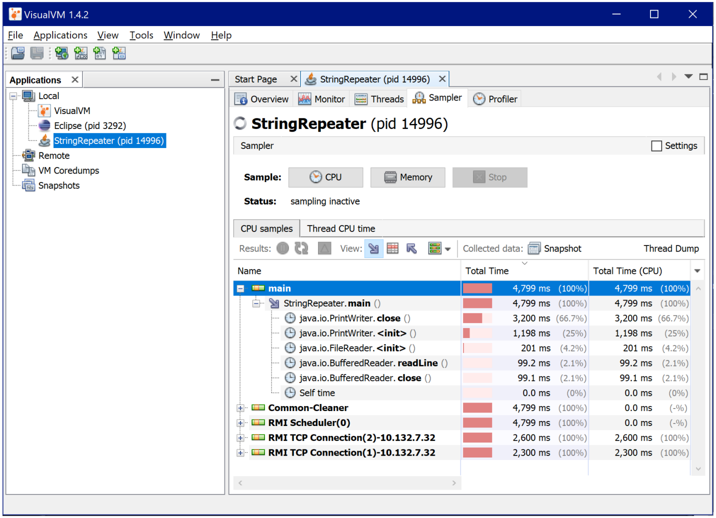


The monitor tab provides a quick, high-level overview of the state of your program. Here, you can see CPU and memory usage in real time.

Sampler / Profiler

The sampler and profiler tab provide access to a sampling profiler and an instrumentation profiler, respectively.
While the instrumentation profiler can be used to collect more accurate, targeted data if examining a specific part of your code, the sampler is easier to use and good enough for our purposes.
Push either the “CPU” or “Memory” button to begin collecting data on runtime or memory usage, respectively. Sampling stops when “Stop” is pushed.
The collected data will be displayed for you to explore.

Sampler / Profiler


VisualVM 1.4.2
File Applications View Tools Window Help
Applications
Local
VisualVM
Eclipse (pid 3292)
StringRepeater (pid 14996)
Remote
VM Coredumps
Snapshots
Start Page
StringRepeater (pid 14996)
Overview Monitor Threads Sampler Profiler
StringRepeater (pid 14996)
Sampler
Settings
Sample: CPU Memory Stop
Status: sampling inactive
CPU samples Thread CPU time
Results: View: Collected data: Snapshot Thread Dump

Name	Total Time		Total Time (CPU)	
main	4,799 ms	(100%)	4,799 ms	(100%)
StringRepeater.**main** ()	4,799 ms	(100%)	4,799 ms	(100%)
java.io.PrintWriter.**close** ()	3,200 ms	(66.7%)	3,200 ms	(66.7%)
java.io.PrintWriter.**<init>** ()	1,198 ms	(25%)	1,198 ms	(25%)
java.io.FileReader.**<init>** ()	201 ms	(4.2%)	201 ms	(4.2%)
java.io.BufferedReader.**readLine** ()	99.2 ms	(2.1%)	99.2 ms	(2.1%)
java.io.BufferedReader.**close** ()	99.1 ms	(2.1%)	99.1 ms	(2.1%)
Self time	0.0 ms	(0%)	0.0 ms	(0%)
Common-Cleaner	4,799 ms	(100%)	0.0 ms	(-%)
RMI Scheduler(0)	4,799 ms	(100%)	0.0 ms	(-%)
RMI TCP Connection(2)-10.132.7.32	2,600 ms	(100%)	2,600 ms	(100%)
RMI TCP Connection(1)-10.132.7.32	2,300 ms	(100%)	2,300 ms	(100%)

Exercise

In the profiling folder, two files are included: `StringRepeater` and `BetterStringRepeater`. One uses a `StringBuilder` to concatenate Strings, and the other uses the concatenation operator. In this part of the lab, you will use VisualVM or `System.nanoTime()` to study the performance of the code.