Optimization and Profiling

CPSC 219: Introduction to Computer Science for Multidisciplinary Studies II
Fall 2023

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Friday, 10 November 2023

Copyright © 2023

Definition

Optimization

- ***Optimization*** is the process of modifying a program to improve its efficiency
  - Increase its speed
  - Reduce its size (memory usage)

- **Optimization can often be seen as de-factoring**
  - **Program gets faster but…**
  - **Harder to understand, upkeep, read**

UNIVERSITY OF CALGARY

Efficiency

- Efficiency can be viewed in terms of:
  1. Program requirements
     - Does the program really need to run at a certain speed? Is it worth the extra effort
  2. Program design
     - If performance is important, design a performance-oriented architecture
       - Set resource goals for individual subsystems and classes
  3. Class and routine design
     - Choose efficient algorithms and datatypes
       - E.g. Quicksort vs. bubble sort
       - E.g. Binary search vs. linear search

UNIVERSITY OF CALGARY

Efficiency (cont'd)

4. Operating system interactions
   - Working with files, dynamic memory, or I/O devices means using system calls
     - May be slow or fast

5. Code compilation
   - Good compilers produce optimized machine code
     - May have options for different optimization levels

UNIVERSITY OF CALGARY

Efficiency (cont'd)

6. Hardware
   - A hardware upgrade may be the cheapest way to improve a program's performance
     - Not always possible

7. Code tuning
   - Small-scale changes made to code to make it run more efficiently
     - At the level of a single routine, or a few lines of code
   - Tends to produce hard-to-understand code
     - Obscures design

UNIVERSITY OF CALGARY

Guide to the galaxy of optimization

General Guidelines

- **Don't** optimize as you go
  - Focusing on optimization during initial development detracts from achieving correctness, readability, and design quality

UNIVERSITY OF CALGARY

General Guidelines (cont’d)

- Jackson’s Rules of Optimization:
  - Rule 1. **Don’t do it.**
  - Rule 2 (for experts only). **Don’t do it yet**—that is, not until you have a perfectly clear and unoptimized solution.
- Code tuning should be done only as a last step
  - Knuth: **Pre-mature optimization is the root of all evil**

UNIVERSITY OF CALGARY

General Guidelines (cont'd)

- Optimize bottlenecks
  - **The 80/20 rule:** 20% of program's routines consume 80% of its execution time
    - Knuth found 4% of a FORTRAN program accounted for over 50% of its run time
  - Spend your time fixing these 'bottlenecks'
    - **Don't waste effort on the other parts**

General Guidelines (cont’d)

- Measure performance when optimizing
    - Use a profiler to find bottlenecks
    - Use timers to measure CPU time
        - Make sure a change actually improves speed
            - May actually make things worse when using a different compiler, OS, or processor

- Run regression tests after each optimization
    - Make sure your program is still correct

UNIVERSITY OF CALGARY

Swipe right

UNIVERSITY OF CALGARY

Profiling

- **Profiling**
  - Is used to find how much time is spent in each function of a program
    - Helps find bottlenecks
    - Helps you compare the performance of algorithms or programs

UNIVERSITY OF CALGARY

Profiling (cont'd)

- Works by sampling the program counter (PC register)
  - Periodically queries the program, recording the function in which it is running
- Is statistical in nature
  - i.e. is somewhat inexact, and will vary from run to run
- Also the act of enabling profiling will generally slow down operation of code, this slowdown can be different for varying classes

UNIVERSITY OF CALGARY

Java profiling

UNIVERSITY OF CALGARY

Profilers – Java

- Standard JVM Profilers
  - VisualVM, JProfiler, YourKit and Java Mission Control
  - method calls and memory usage
  - **Pros:**
    - Great for tracking down memory leaks, standard profilers detail out all memory usage by the JVM and which classes/objects are responsible.
    - Good for tracking CPU usage and zero in on hot spots.

UNIVERSITY OF CALGARY

Profilers – Java

- Standard JVM Profilers
  - VisualVM, JProfiler, YourKit and Java Mission Control
  - method calls and memory usage
  - **Cons:**
    - Requires a direct connection to the monitored JVM; this ends up limiting usage to development environments in most cases.
    - They slow down your application; a good deal of processing power is required for the high level of detail provided.

UNIVERSITY OF CALGARY

Profilers – Java

- Lightweight Java Transaction Profilers
  - XRebel and Stackify Prefix
  - Aspect Profilers
    - use aspect-oriented programming (AOP) to inject code into the start and end of specified methods.
  - Java Agent profilers (ex. Netbeans built-in)
    - use the Java Instrumentation API to inject code into your application. This method has greater access to your application since the code is being rewritten at the bytecode level.

Profilers – Java

- Lightweight Java Transaction Profilers
  - Aspect profilers are pretty easy to setup but are limited in what they can monitor and are encumbered by detailing out everything you want to be tracked.
  - Java Agents have a big advantage in their tracking depth but are much more complicated to write.

UNIVERSITY OF CALGARY

Profilers – Java

- Low Overhead, Java JVM Profiling in Production (APM – APplication Monitoring)
  - New Relic, AppDynamics, Stackify Retrace, Dynatrace
  - how your system performs in production is critical
  - Java APM tools typically use the Java Agent profiler method
    - different instrumentation rules to allow them to run without affecting performance in productions.

Memory

UNIVERSITY OF CALGARY

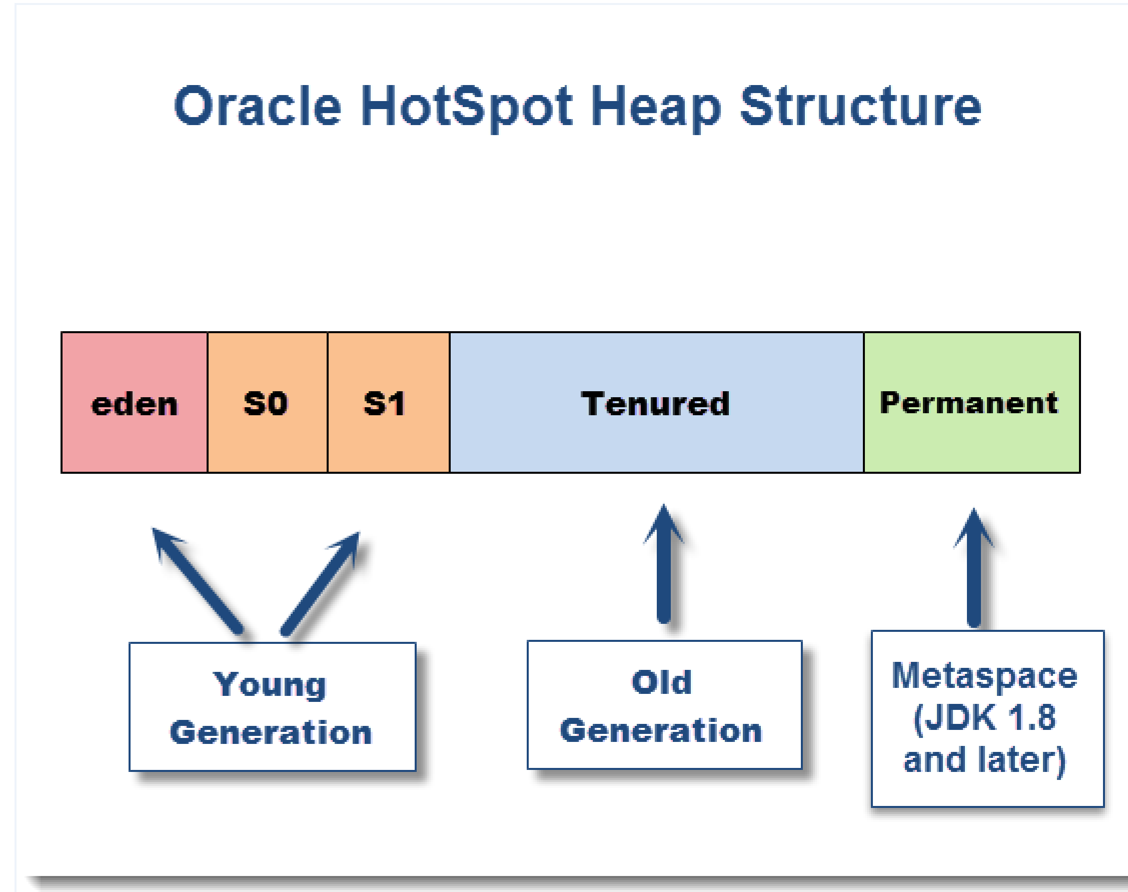
Code Tuning – Memory Leaks

- Java is stuck with garbage collection
- We can stop point at things but not delete them
- If your program naively leaves created objects connected to current code (heap will continue to grow)
- You can generally see this via Profiling and heap dumps

UNIVERSITY OF CALGARY

Code Tuning – Heap Structure

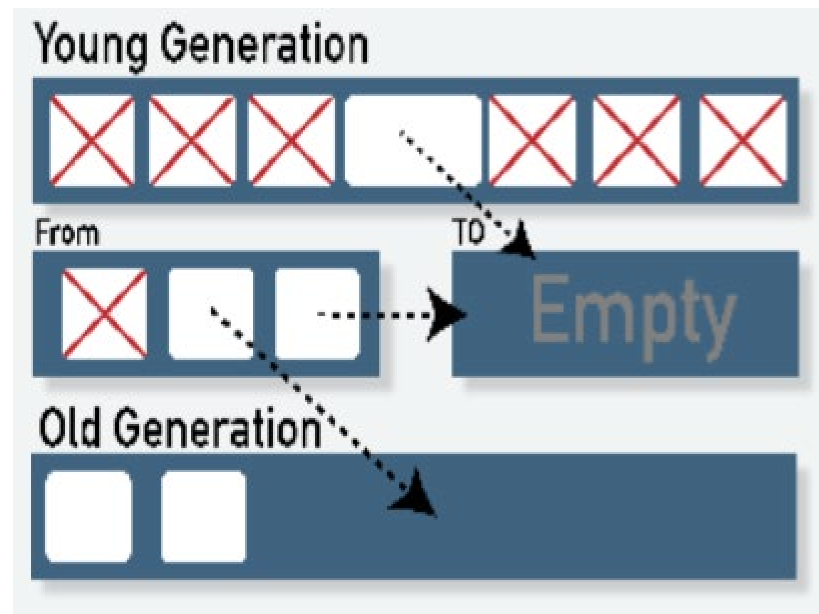
- The young generation is actually garbage collected quicker than the older generation
- Lots of new objects, or aggressive GC in young generation slows down program



UNIVERSITY OF CALGARY

Code Tuning – Garbage Collectors

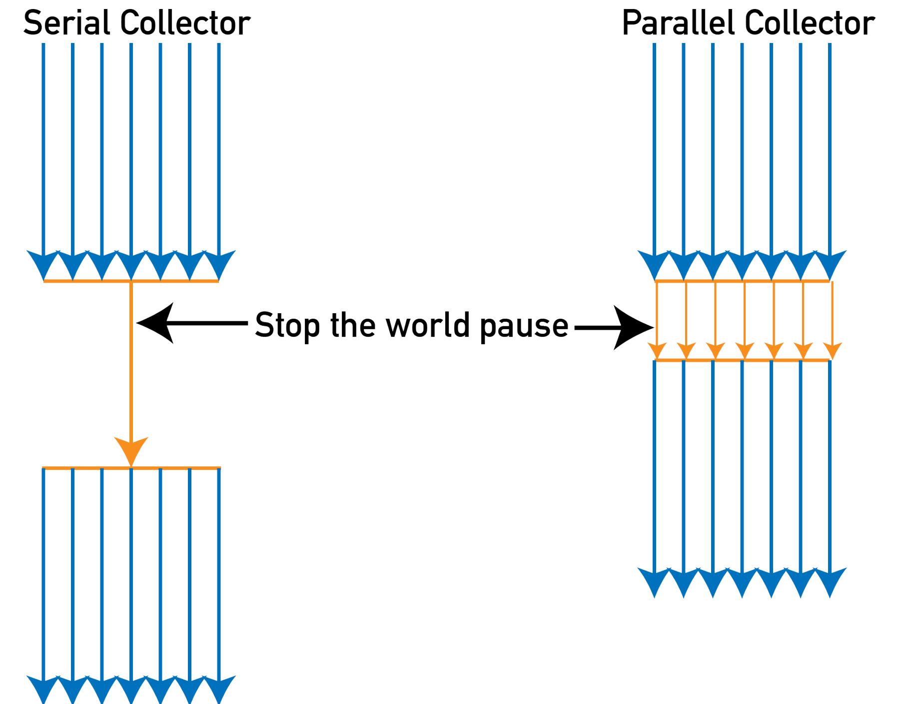
- Serial Collector
  - Both Young and Old collections are done serially, using a single CPU and in a stop-the-world fashion.
  - Best client-side



UNIVERSITY OF CALGARY

Code Tuning – Garbage Collectors

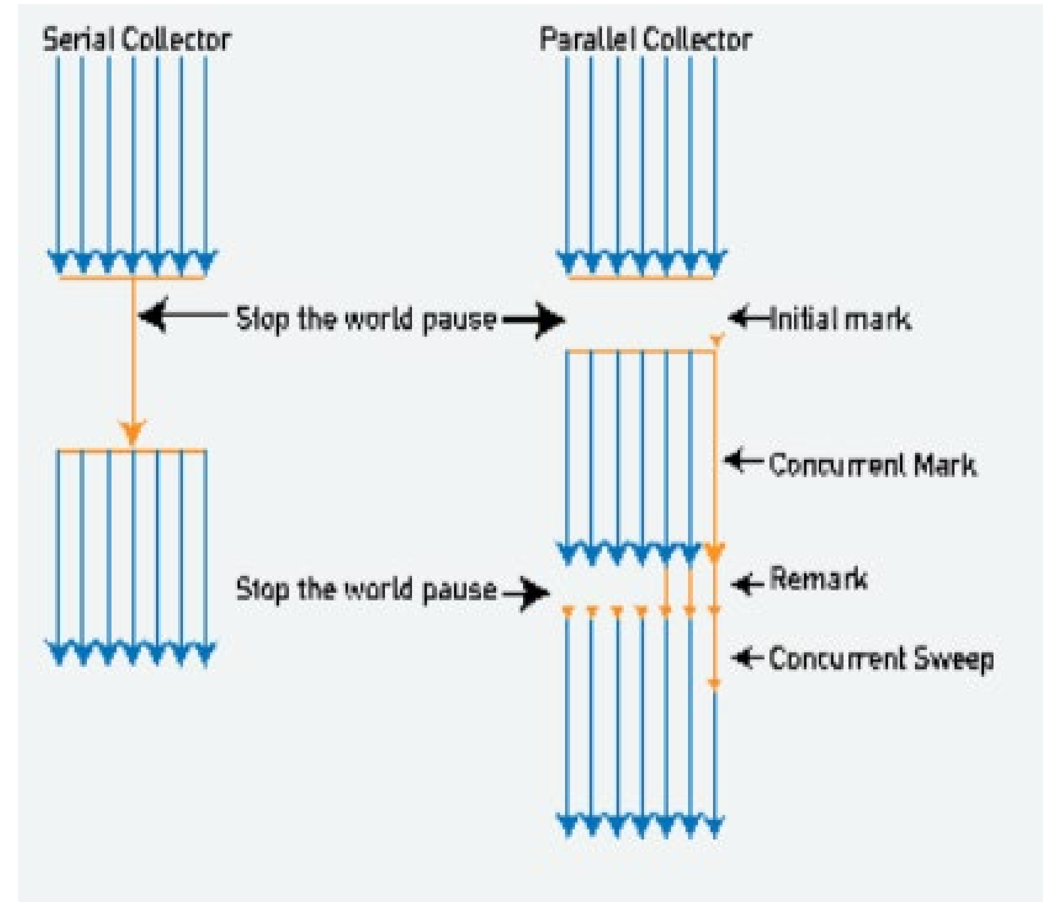
- Serial Collector
  - Both Young and Old collections are done serially, using a single CPU and in a stop-the-world fashion.
  - Best client-side
- Parallel Collector(throughput collector)
  - Designed to take advantage of available CPU cores. Both Young and Old collections are done using multiple Gcthreads.



UNIVERSITY OF CALGARY

Code Tuning – Garbage Collectors

- Mostly concurrent collectors (low-latency collectors)
  - Designed to minimize impact on application response time associated with Old generation stop-the-world collections.
  - Most of the collection of the old generation using the CMS collector is done concurrently with the execution of the application.



UNIVERSITY OF CALGARY

Code Tuning – Garbage Collectors

- Choose wisely between 32-bit or 64-bit VMs

  - going from a 32-bit to a 64-bit machine increases heap requirement for an existing Java application by up to 1.5 times (bigger ordinary object pointers)

  - -XX:+UseCompressedOops in Java version prior to 1.7 (which is now default)
    - This tuning argument greatly alleviates the performance penalty associated with a 64-bit JVM.

UNIVERSITY OF CALGARY

Code Tuning – Garbage Collectors

- Large heap not always better
    - Profile your application for possible memory leaks using tools such as Java VisualVM or Plumbr (Java memory leak detector).
    - Focus your analysis on the biggest Java object accumulation points
    - Reducing your application memory footprint will translate in improved performance due to reduced GC activity.

UNIVERSITY OF CALGARY

Algorithm Based Optimization

UNIVERSITY OF CALGARY

Algorithm-Based Optimization

- Choosing a more efficient algorithm or data structure is often the best way to improve program efficiency

  - Look for algorithms that reduce the order of complexity

    - E.g. Binary search $O(\log n)$ vs. linear search $O(n)$
    - E.g. Merge sort $O(n\log n)$ vs. bubble sort $O(n^2)$

UNIVERSITY OF CALGARY

Algorithm-Based Optimization

- Do this first before attempting other optimizations
  - Hand tuning an $O(n^2)$ algorithm won't yield near the same gains as using an $O(n \log n)$ algorithm

- **Beware of worst-case performance**
  - Some algorithms may not achieve their average Big-O performance under certain conditions
  - E.g. The quicksort degenerates to $O(n^2)$ with nearly-sorted inputs

UNIVERSITY OF CALGARY

Algorithm-Based Optimization

- Sometimes an inefficient algorithm is fine for small inputs
    - The overhead of a complicated algorithm may make it slower than a simple one
        - And harder to debug and maintain!
    - Measure performance to make sure you've made the right choice

UNIVERSITY OF CALGARY

Algorithm-Based Optimization

- Sometimes an inefficient algorithm is fine for small inputs
  - **Java's own internal Quick Sort uses an Insertion Sort below a specific array size**

UNIVERSITY OF CALGARY

Compiler Based Optimization

UNIVERSITY OF CALGARY

Compiler-Level Optimization

- Enabling compiler optimization can improve speed by as much as 2 times
- Most compilers turn off optimization by default
    - Optimized code tends to confuse debuggers
- Works best with straightforward code
    - Hand tuned code may actually be harder for the compiler to optimize

UNIVERSITY OF CALGARY

Java Specific Optimizations

UNIVERSITY OF CALGARY

Code Tuning – Java

- Java is an object oriented language
- That runs in a virtual machine
- There are more inefficiencies that can be improved than we’ve covered for a language like c++

UNIVERSITY OF CALGARY

Strings

UNIVERSITY OF CALGARY

Code Tuning – Strings

- Not null terminated
  - char[] and length are both stored

- Immutable
  - Any change attempt (making new string)

- Also UTF-16 (uses two bytes for all)
  - if you want UTF-32 there's a lot of management steps

UNIVERSITY OF CALGARY

Code Tuning – Strings

- **String pool**
  - Java has a special memory location (PermGen Space)
    - Usually for things like class desc, and metadata (exist longterm)
  - If a new String literal (“hello”) is made matching existing Java will attempt to point at same data
    - No NEW object
  - new String(“hello”) by-passes this
  - Also dynamic strings like one created at runtime from input won’t be associated

UNIVERSITY OF CALGARY

Code Tuning – Strings

- **String pool**
  - Java has a special memory location (PermGen Space)
    - Usually for things like class desc, and metadata (exist longterm)

```
public static void main(String[] args){
    System.out.println(System.identityHashCode("hello"));
    System.out.println(System.identityHashCode("hello"));
    System.out.println(System.identityHashCode(new String("hello")));
}
```

- 366712642
- 366712642
- 1829164700

UNIVERSITY OF CALGARY

Code Tuning – Strings

```
Scanner s;
s = new Scanner(System.in);
System.out.println(System.identityHashCode("hello"));
System.out.println(System.identityHashCode("hello"));
System.out.println(System.identityHashCode(new String("hello")));
String str = s.nextLine();
str = str.trim();
System.out.println(System.identityHashCode(str));
```

- 1442407170
- 1442407170
- 1028566121
- hello
- 1118140819

UNIVERSITY OF CALGARY

Code Tuning – Strings

- **String pool**
  - Java has a special memory location (PermGen Space)
    - Usually for things like class desc, and metadata (exist longterm)
  - USE .equals()
    - To get consistent String comparisons on .equals() compares contents, == will give you differing behaviour whether or not the String Pool has been used

UNIVERSITY OF CALGARY

Code Tuning – Strings

- **String pool**
  - USE .equals()
    - To get consistent String comparisons on .equals() compares contents, == will give you differing behaviour whether or not the String Pool has been used
  - Example: Junit Testing
    - Setup will contain string literals String pool which re-use memory, thus == will work
    - however during operation == may fail
    - Strings during operation often collected via input steps

UNIVERSITY OF CALGARY

Code Tuning – Strings

- StringBuilder and StringBuffer
  - StringBuilder not thread-safe

- Let you compile a list of Strings which you can convert to a final String once
  - Much better than repetitive +, += operations

- Can even set expected capacity needed (like ArrayList) so that hidden array doesn’t need to expand

Maps

UNIVERSITY OF CALGARY

Code Tuning – Maps

- When you want to iterate through a Map, and you need both keys and values, instead of the following:

```
for (K key : map.keySet()) {
    V value : map.get(key);
}
```

- .. To this:

```
for (Entry<K, V> entry : map.entrySet()) {
    K key = entry.getKey();
    V value = entry.getValue();
}
```


UNIVERSITY OF CALGARY

Code Tuning – hashCode()/equals()

- Optimise your hashCode() and equals() methods
- A good hashCode() method is essential because it will prevent further calls to the much more expensive equals()
- Can store a calculated hashCode once in object (only update on modified object, when sets are called)

UNIVERSITY OF CALGARY

Primitives

UNIVERSITY OF CALGARY

Code Tuning – Primitives

- Reverse of refactoring
- Sometimes code tuning is called ‘defactoring’
- Use double instead of Double, int instead of Integer
- Java can store values on stack, instead of heap
- Try to avoid BigInteger and BigDecimal, similarly
  - Only if you really need to exceed long, or need precision

UNIVERSITY OF CALGARY

Logging

UNIVERSITY OF CALGARY

Code Tuning – Logging

- Strings take a lot of time to create (program-wise)
- Check the current log level first before making log string

```
// don’t do this
log.debug(“User [” + userName + “] called method X with [” + i + “]”);

// or this
log.debug(String.format(“User [%s] called method X with [%d]”,
userName, i));

// do this
if (log.isDebugEnabled()) {
log.debug(“User [” + userName + “] called method X with [” + i + “]”);
}
```


UNIVERSITY OF CALGARY

Libraries

Code Tuning – Libraries

- Use Apache Commons StringUtils.replace instead of String.replace
  - Java 9 improved String replace but if on Java 8

```
// replace this
test.replace(“test”, “simple test”);

// with this
StringUtils.replace(test, “test”, “simple test”);
```


UNIVERSITY OF CALGARY

Code Tuning – Libraries

- Avoid regular expressions and instead use Apache Commons Lang.

UNIVERSITY OF CALGARY

Simple Recursion

UNIVERSITY OF CALGARY

Code Tuning – Recursion

- Recursion is great for design of algorithms but not great for optimization

- Stay away from recursion.
  - **Recursion is very resource intensive!**

- Very beneficial to code tune algorithms to be loops instead of recursive calls
  - Replace program stack with self-managed stack structure for data that would normally be passed in recursive call

UNIVERSITY OF CALGARY

Code Tuning – Recursion

```
public void countDown(int n) {
    if (n == 0) {
        return;
    }
    System.out.println(n + "...");
    waitASecond();
    countDown(n - 1);
}

public void countDown(int n) {
    while (n > 0) {
        System.out.println(n + "...");
        waitASecond();
        n -= 1;
    }
}
```


UNIVERSITY OF CALGARY

Caching

UNIVERSITY OF CALGARY

Code Tuning – Hidden Caching

- A typical example is caching database connections in a pool.
  - The creation of a new connection takes time, which you can avoid if you reuse an existing connection.

- You can also find other examples in the Java language itself.
  - The valueOf method of the Integer class, for example, caches the values between -128 and 127.

UNIVERSITY OF CALGARY

Iterators

UNIVERSITY OF CALGARY

Code Tuning – Iterators

- Common now to use Java iterators
  - Is a good refactoring, but depending…
  - for (String value: strings) { // Do something useful here }

- a new iterator instance will be created

```
int size = strings.size();
for (int i = 0; i < size; i++) {
    String value: strings.get(i);
    // Do something useful here
}
```


UNIVERSITY OF CALGARY

Onward to … CPSC 331 Data Structures.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/