Optimization: Java Optimization

CPSC 501: Advanced Programming Techniques
Winter 2025

Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Wednesday, March 5, 2025

Java Specific Optimizations

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++

Strings

Code Tuning – Strings

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

- Immutable
  - Any change attempt (making new string)

- char[] is better for secure data than String

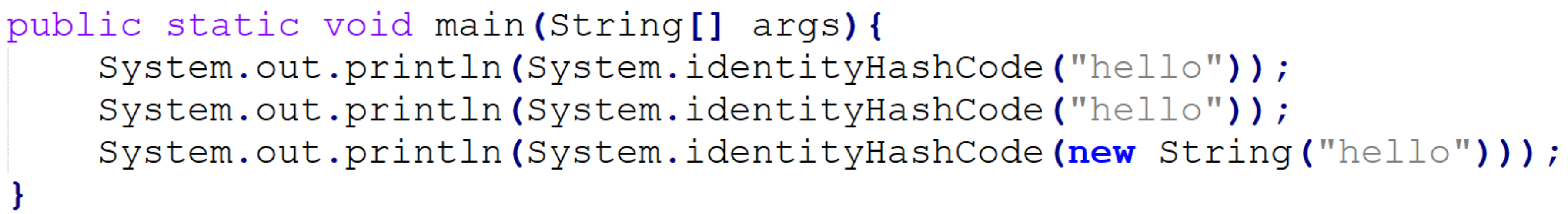
- Also UTF-16 (uses two bytes for all)
  - if you want UTF-32 there’s a lot of management steps

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

Code Tuning – Strings

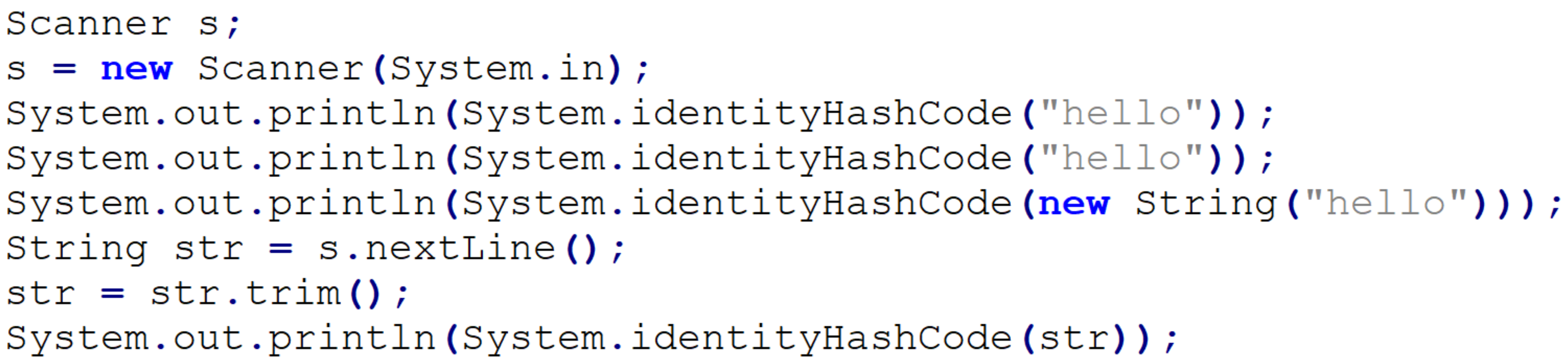
- **String pool**
  - Java has a special memory location (PermGen Space)
    - Usually for things like class desc, and metadata (exist longterm)



```
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

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

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

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

Code Tuning – Strings

- StringBuilder and StringBuffer
  - StringBuilder not thread-safe

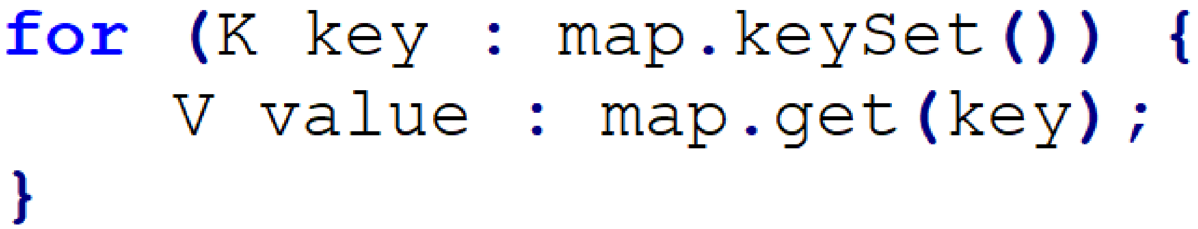
- Let you compile a list of Strings which you can convert to a final String once
  - Much better than repetitive +, += operations

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

Maps

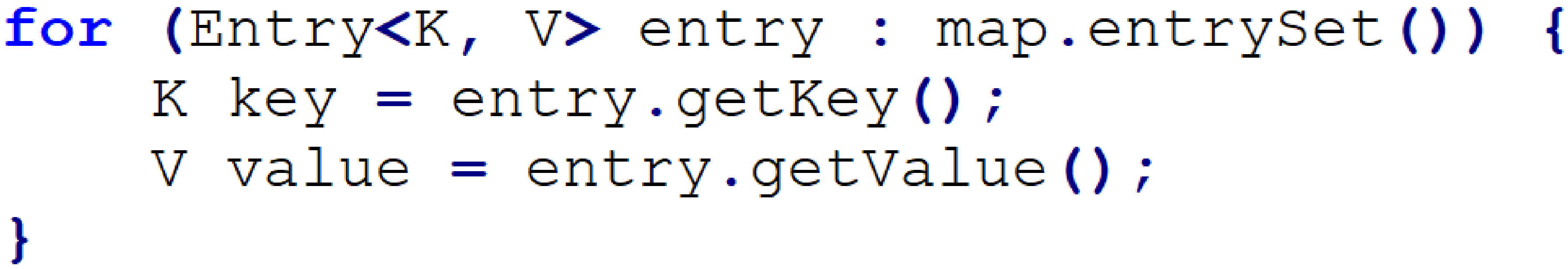
Code Tuning – Maps

- 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();
}
```

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)

Primitives

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

Code Tuning – Primitives

- Try to avoid BigInteger and BigDecimal, similarly
  - Only if you really need to exceed long, or need precision
  - int > Integer > BigItenger   double > Double > BigDecimal

- Integer is not a primitive (it is Object and is immutable)

  - x = new Integer(1) , x = x + new Integer(2), x = new Integer(3)
    - 1,2,3 are all individual objects and x is ‘pointed’ towards a new one

  - x = 1 , x = x + 2, x = 3 , the memory spot x points to is changed from 1 to 3

Logging

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 + “]”);
}
```

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”);
```

Code Tuning – Libraries

- Avoid regular expressions and instead use Apache Commons Lang.

Simple Recursion

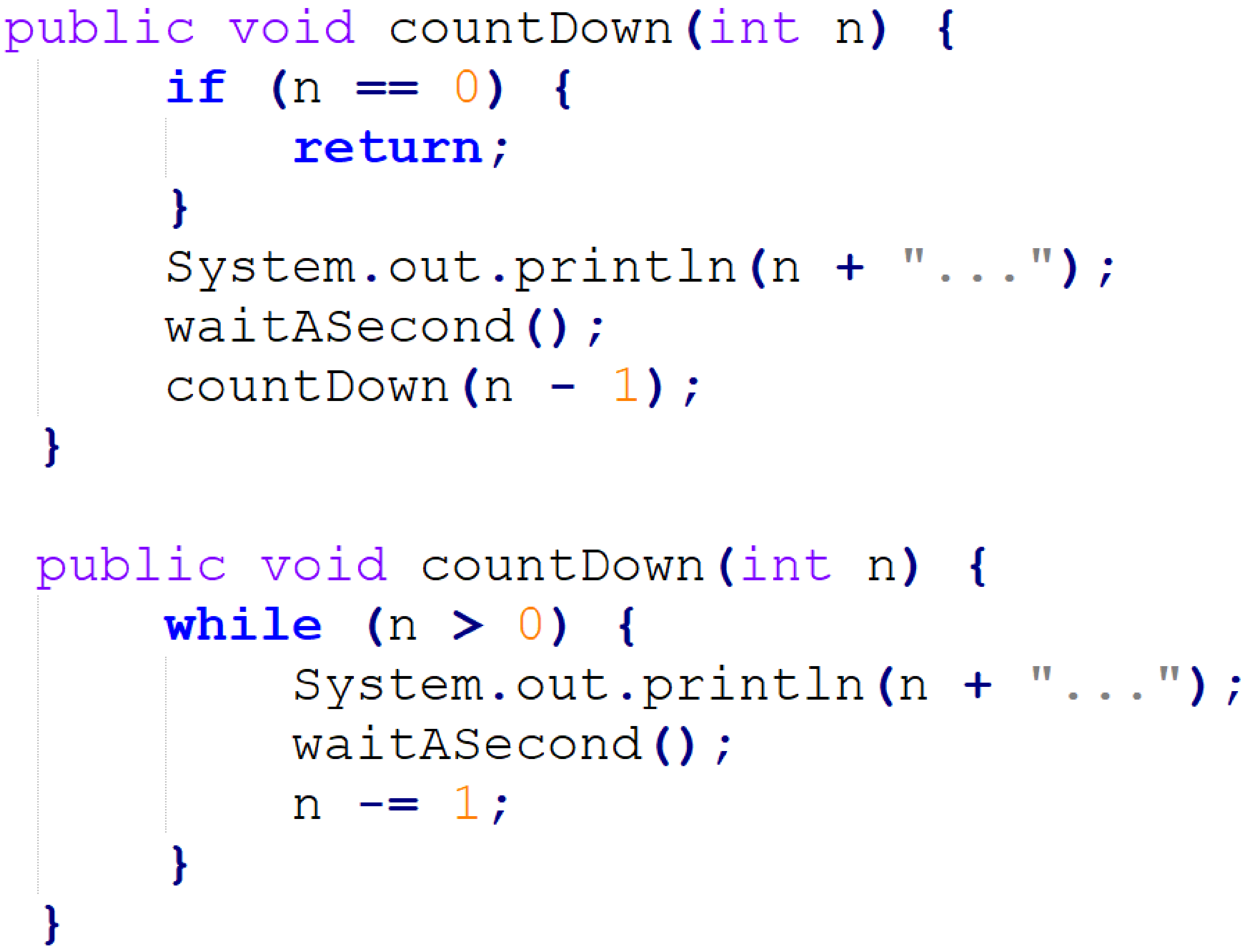
Code Tuning – Recursion

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

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

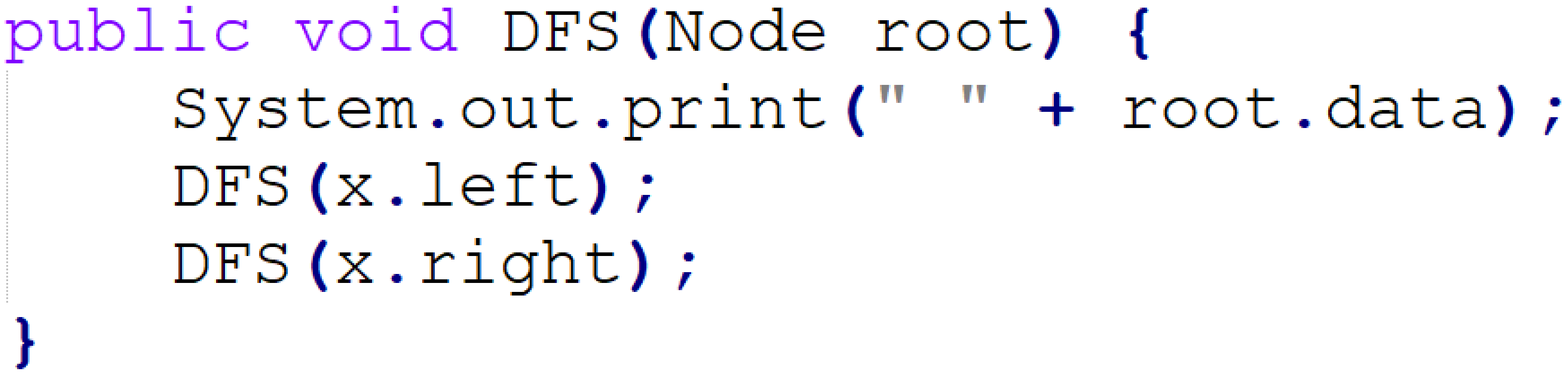
- Very beneficial to code tune algorithms to be loops instead of recursive calls
  - Replace program stack with self-managed stack structure for data that would normally be passed in recursive call

Code Tuning – Recursion



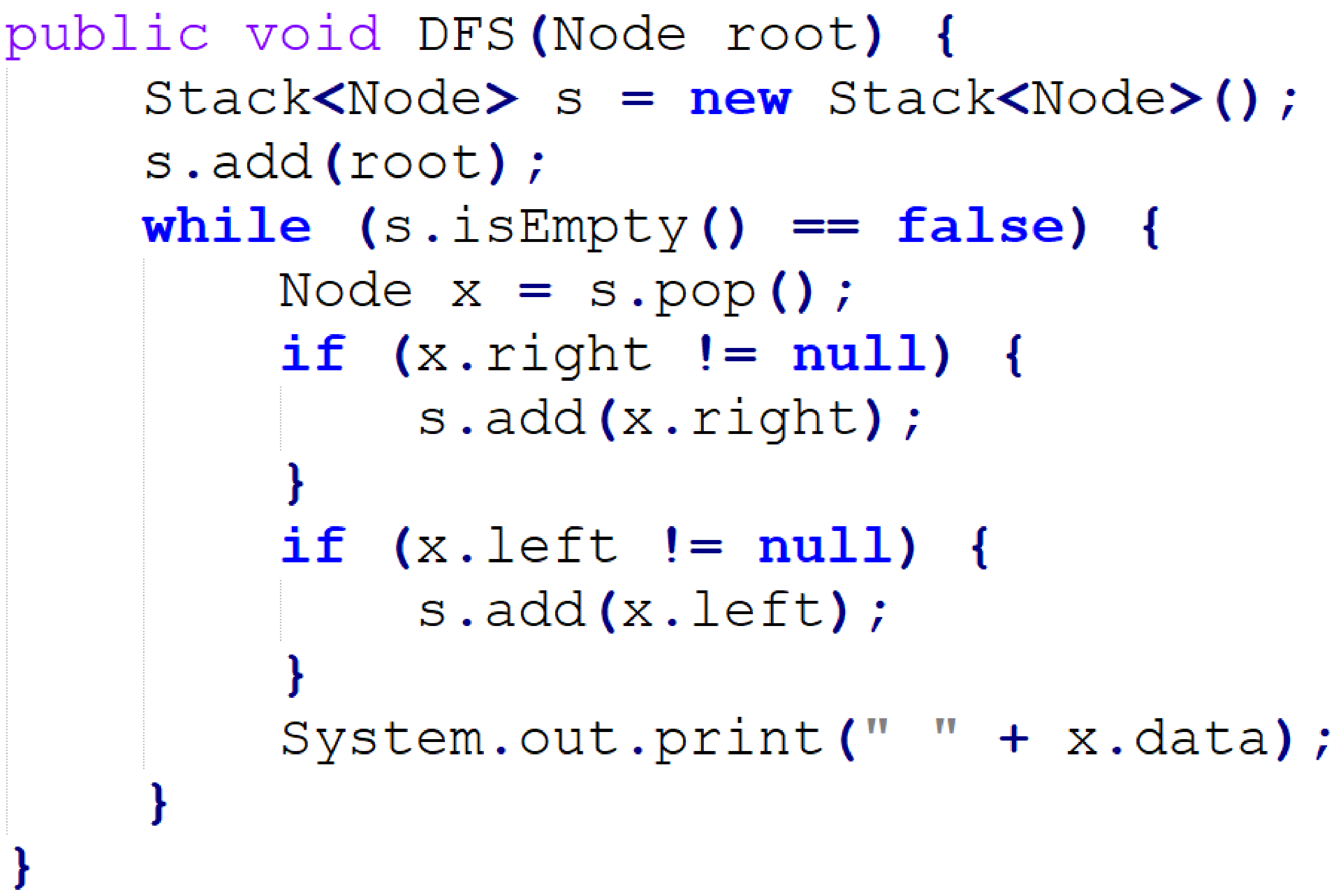
```
public void countDown(int n) {
     if (n == 0) {
         return;
     }
     System.out.println(n + "...");
     waitASecond();
     countDown(n - 1);
 }

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

Code Tuning – Recursion



```
public void DFS(Node root) {
    System.out.print(" " + root.data);
    DFS(x.left);
    DFS(x.right);
}
```

Code Tuning – Recursion



```java
public void DFS(Node root) {
    Stack<Node> s = new Stack<Node>();
    s.add(root);
    while (s.isEmpty() == false) {
        Node x = s.pop();
        if (x.right != null) {
            s.add(x.right);
        }
        if (x.left != null) {
            s.add(x.left);
        }
        System.out.print(" " + x.data);
    }
}
```

Caching

Code Tuning – Hidden Caching/Pooling

- 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.

Iterators

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
}
```

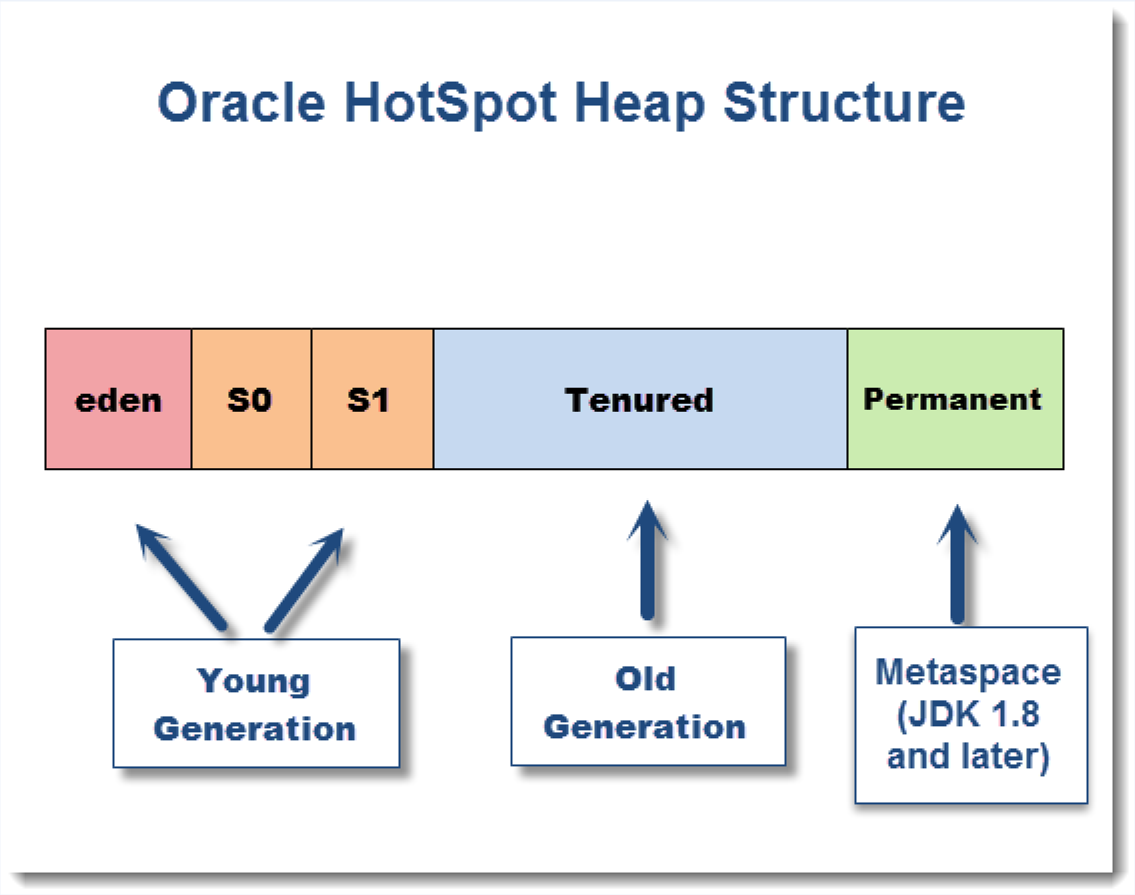
Memory

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

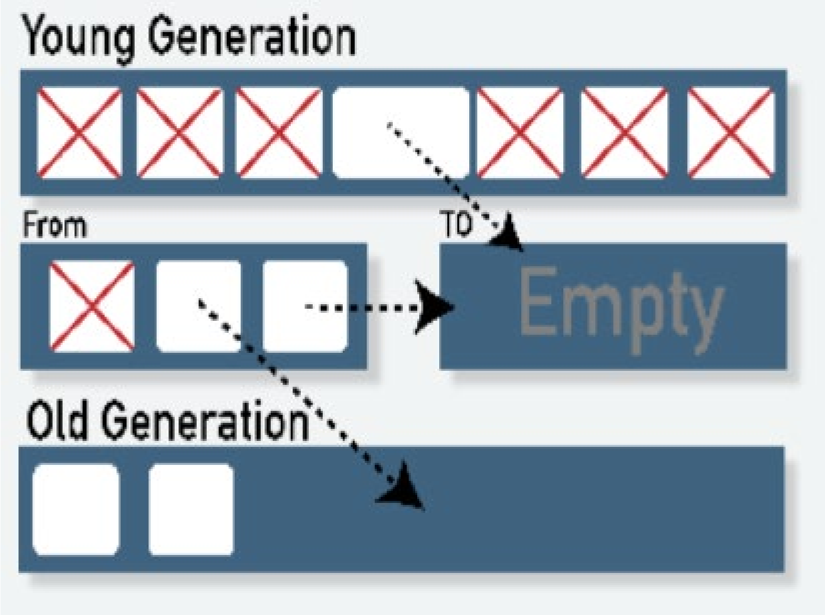
Code Tuning – Heap Structure

- 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



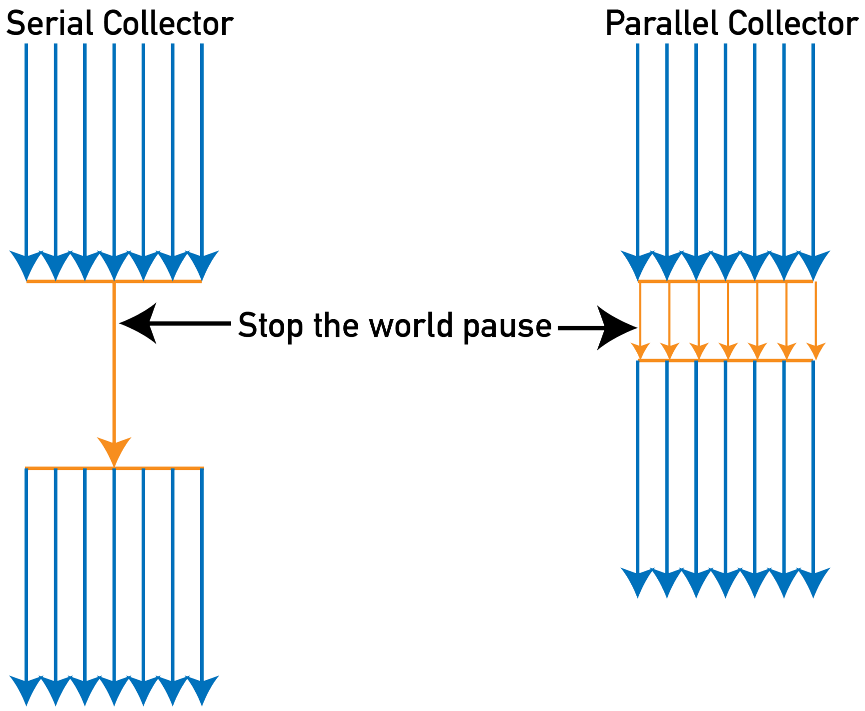
Code Tuning – Garbage Collectors

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



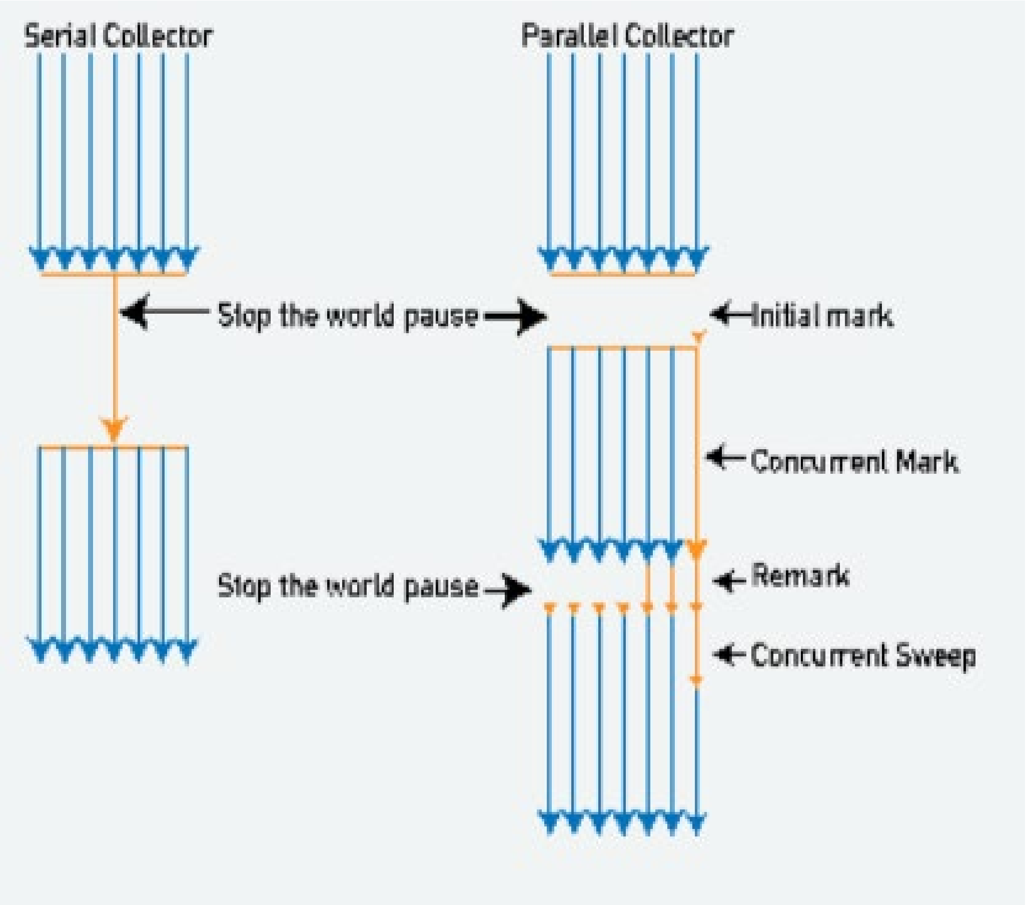
Code Tuning – Garbage Collectors

- 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.



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.



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.

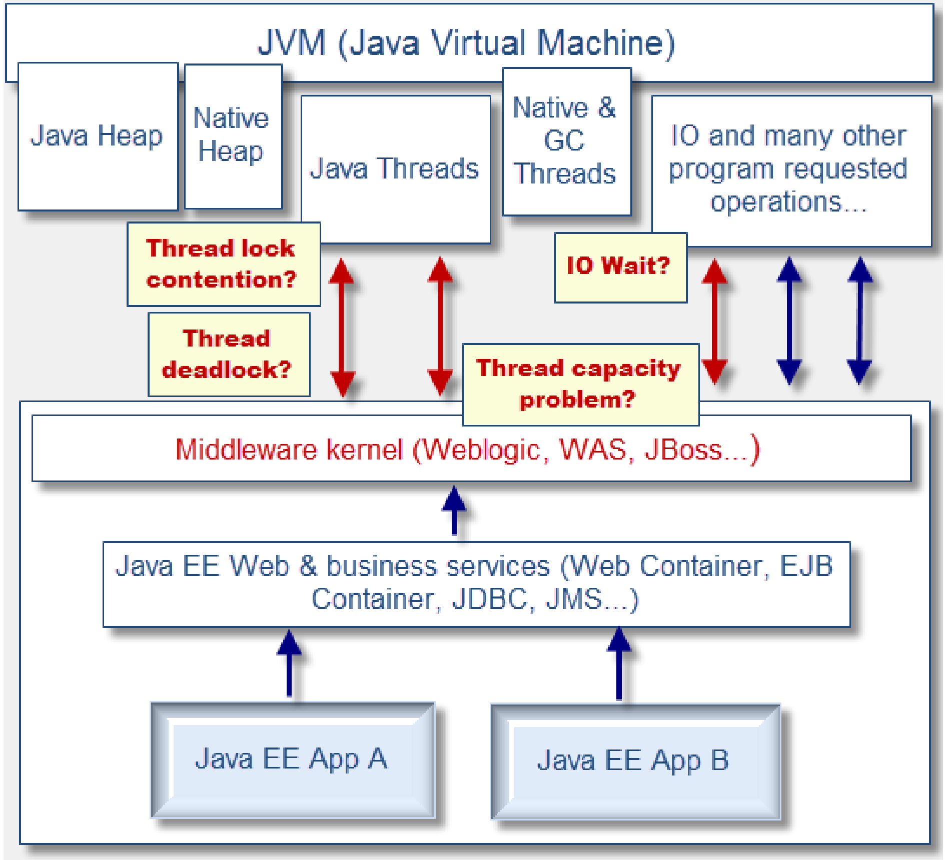
Code Tuning – Garbage Collectors

- 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.

Threads

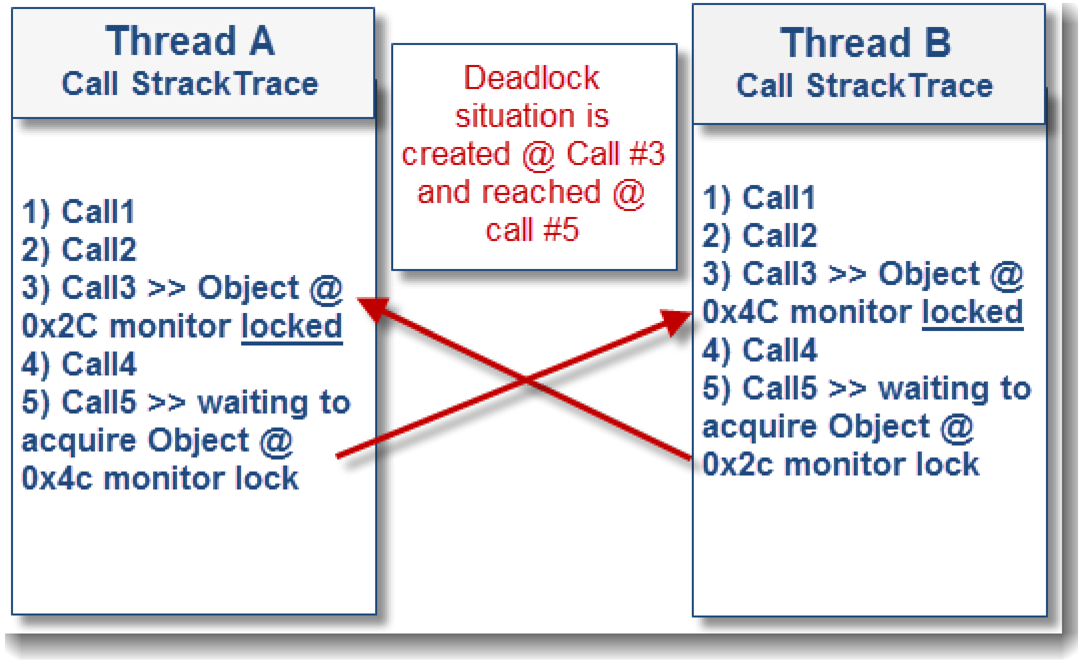
Code Tuning – Thread-Lock/Contention

- Thread lock contention is by far the most common Java concurrency problem


JVM (Java Virtual Machine)
Java Heap
Native Heap
Java Threads
Native & GC Threads
IO and many other program requested operations…
Thread lock contention?
IO Wait?
Thread deadlock?
Thread capacity problem?
Middleware kernel (Weblogic, WAS, JBoss…)
Java EE Web & business services (Web Container, EJB Container, JDBC, JMS…)
Java EE App A
Java EE App B

Code Tuning – Thread-Lock/Contention

- True Java-level deadlocks, while less common, are triggered when two or more threads are blocked forever, waiting for each other.



Thread A
Call StrackTrace

1) Call1
2) Call2
3) Call3 >> Object @ 0x2C monitor <u>locked</u>
4) Call4
5) Call5 >> waiting to acquire Object @ 0x4c monitor lock

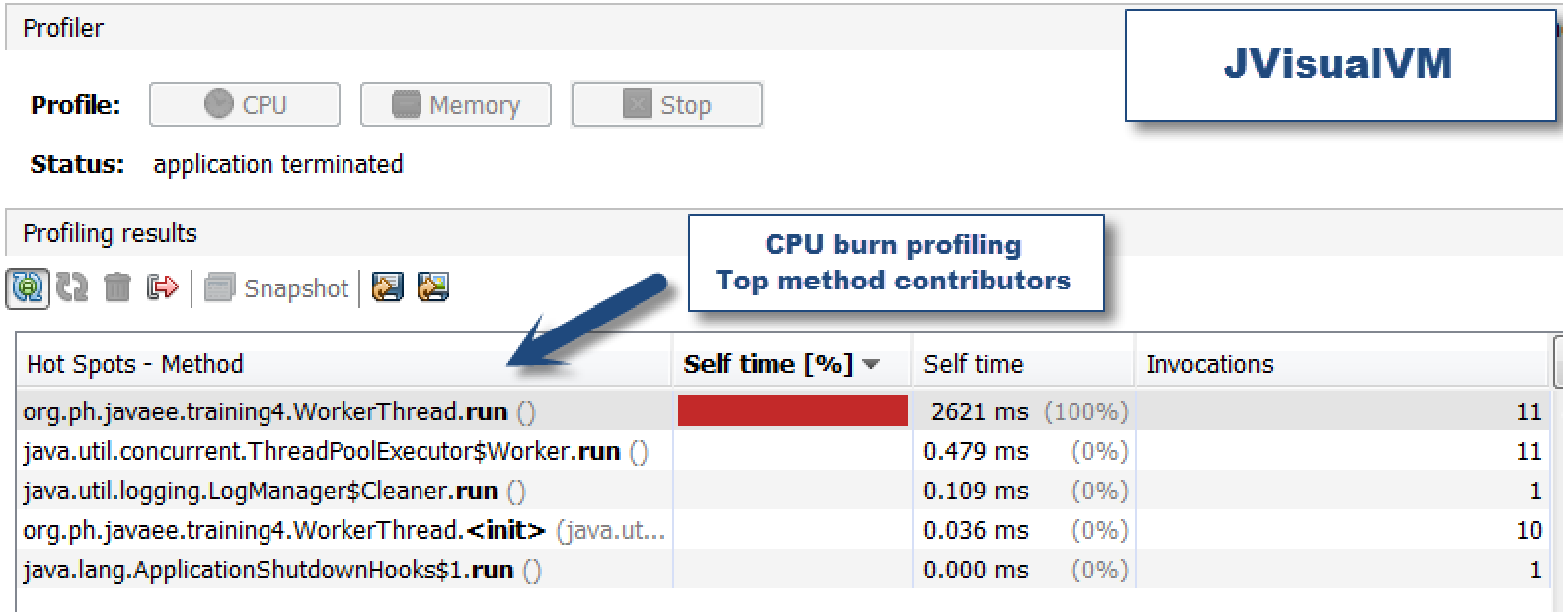
Deadlock situation is created @ Call #3 and reached @ call #5

Thread B
Call StrackTrace

1) Call1
2) Call2
3) Call3 >> Object @ 0x4C monitor <u>locked</u>
4) Call4
5) Call5 >> waiting to acquire Object @ 0x2c monitor lock

Code Tuning – Thread-Lock/Contention

- Clock Time and CPU Burn
  - Ex. worker not doing anything, just spinning in a loop



Status: application terminated

JVisualVM

CPU burn profiling
Top method contributors

Hot Spots - Method	Self time [%]	Self time	Invocations
org.ph.javaee.training4.WorkerThread.**run** ()	🟥	2621 ms (100%)	11
java.util.concurrent.ThreadPoolExecutor$Worker.**run** ()		0.479 ms (0%)	11
java.util.logging.LogManager$Cleaner.**run** ()		0.109 ms (0%)	1
org.ph.javaee.training4.WorkerThread.**<init>** (java.ut...		0.036 ms (0%)	10
java.lang.ApplicationShutdownHooks$1.**run** ()		0.000 ms (0%)	1

Timeout Management

Code Tuning – Timeout Management

- Lack of proper HTTP/HTTPS/TCP IP timeouts between your Java application and external systems
  - lead to severe performance degradation and outage due to middleware and JVM threads depletion (blocking IO calls).

- Proper timeout implementation will prevent Java threads from waiting for too long in the event of major slowdown of your external service providers.

Onward to …
python optimization.

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

UNIVERSITY OF CALGARY