Evaluating the Java Monitor Object Motivating Example



Douglas C. Schmidt <u>d.schmidt@vanderbilt.edu</u> www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems Vanderbilt University Nashville, Tennessee, USA



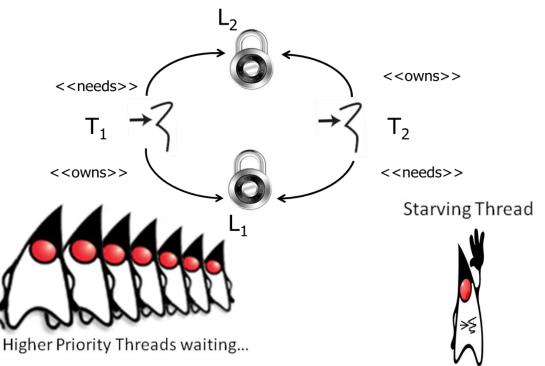
Learning Objectives in this Part of the Lesson

- Understand what monitors are & know how Java built-in monitor objects can
 ensure mutual exclusion & coordination between threads
- Note a human-known use of monitors
- Recognize common synchronization problems in concurrent Java programs using the BuggyQueue case study app
- Be aware of common complexities in concurrent programs like BuggyQueue



Running Java Thread

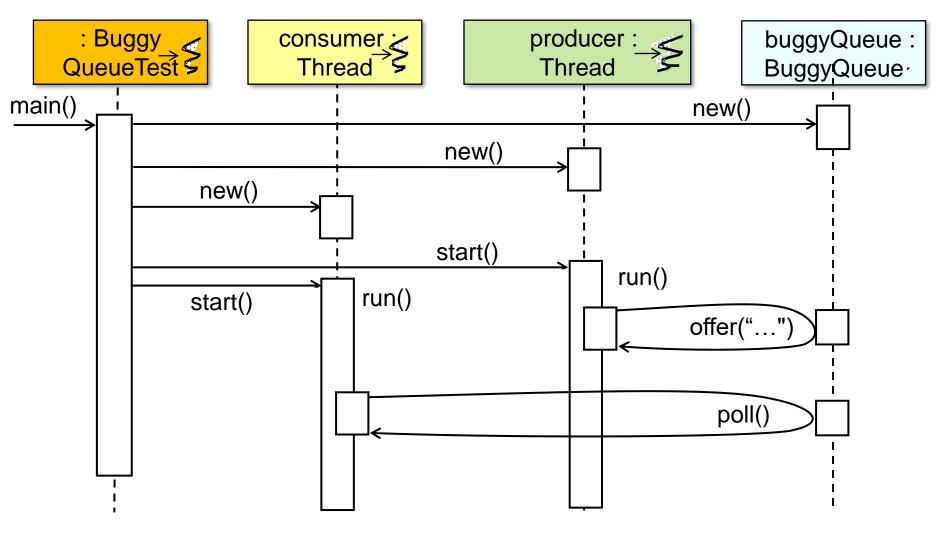




Evaluating the Buggy Producer/Consumer

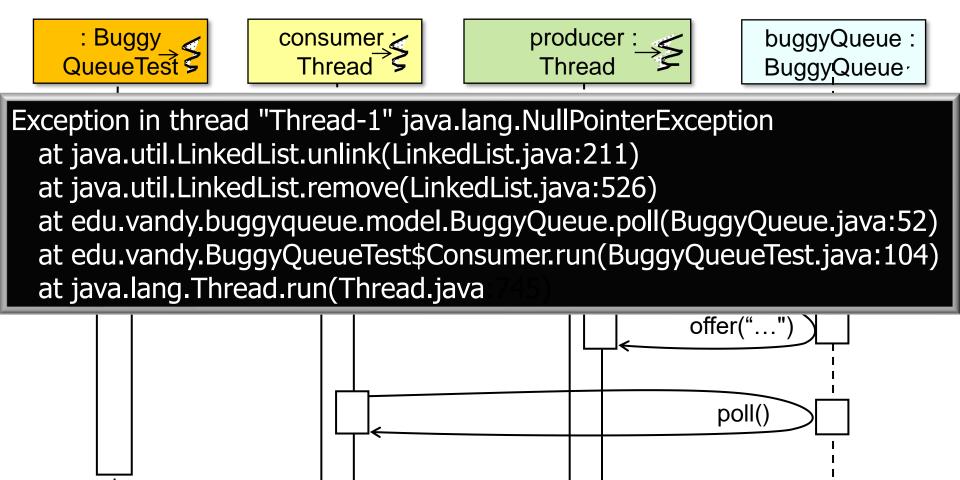
Evaluating the Buggy Producer/Consumer

• Key question: what's the output & why?



Evaluating the Buggy Producer/Consumer

• Key question: what's the output & why?



Depending on the implementation of the BuggyQueue class & the underlying LinkedList the app & test program may simply "hang"

Evaluating the Buggy Producer/Consumer

• Key question: what's the output & why?

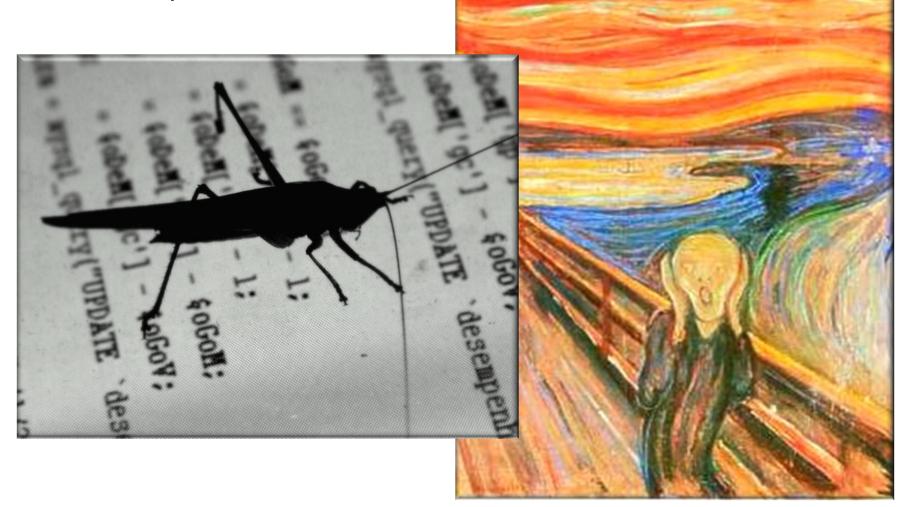
static class BuggyQueue<E> implements BoundedQueue<E> {
 private LinkedList<E> mList = new LinkedList<E>();

```
public boolean offer(E e) {
    if (!isFull()) { mList.add(e); return true; }
    else return false;
    There's no protection against
    critical sections being run by
    multiple threads concurrently
    public E poll() {
        if (!isEmpty()) return mList.remove(0); else return false; }
        ...
}
```

Note that this implementation is not synchronized. If multiple threads access a linked list concurrently, and at least one of the threads modifies the list structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more elements; merely setting the value of an element is not a structural modification.)

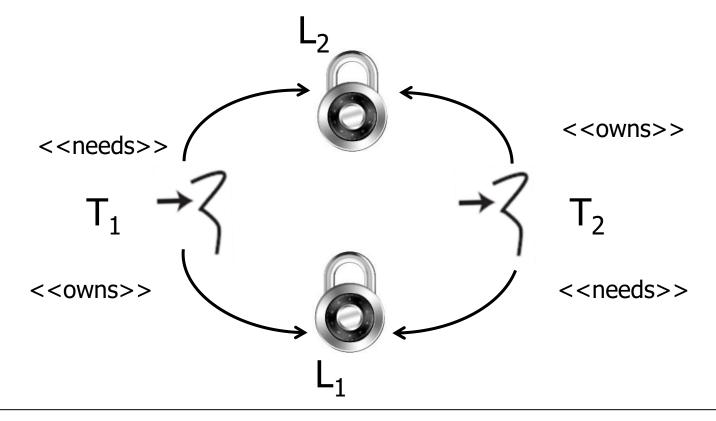
See <u>docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html</u>

Concurrent programs are hard to develop & debug, due to various inherent & accidental complexities



See <u>stackoverflow.com/questions/499634/how</u> <u>-to-detect-and-debug-multi-threading-problems</u>

- Concurrent programs are hard to develop & debug, due to various inherent & accidental complexities, e.g.
 - Deadlock
 - Occurs when two or more competing actions are each waiting for the other to finish, & thus none ever do



See <u>en.wikipedia.org/wiki/Deadlock</u>

- Concurrent programs are hard to develop & debug, due to various inherent & accidental complexities, e.g.
 - Deadlock
 - Starvation
 - A thread is perpetually denied necessary resources to process its work

Running Java Thread





Higher Priority Threads waiting...

Starving Thread



See <u>en.wikipedia.org/wiki/Starvation (computer_science)</u>

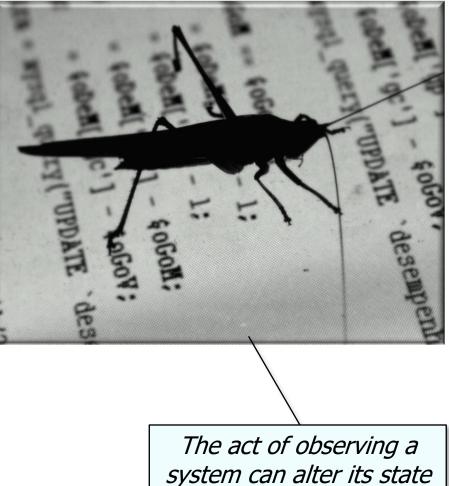
- Concurrent programs are hard to develop & debug, due to various inherent & accidental complexities, e.g.
 - Deadlock
 - Starvation
 - Race conditions
 - Arise when an application depends on the sequence or timing of threads for it to operate properly



See en.wikipedia.org/wiki/Race_condition

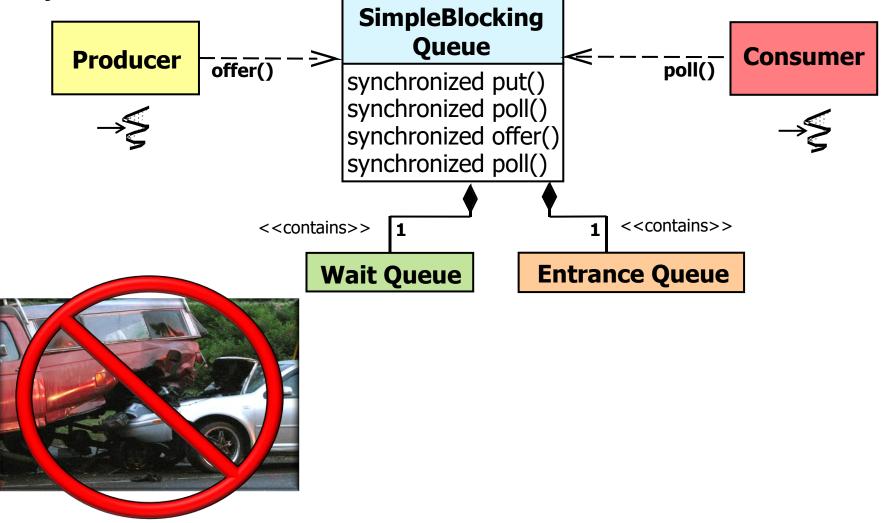
- Concurrent programs are hard to develop & debug, due to various inherent & accidental complexities, e.g.
 - Deadlock
 - Starvation
 - Race conditions
 - Tool limitations
 - e.g., behavior in the debugger doesn't reflect actual behavior

D	60 0 to to . 0. 9				DDM	5 Debur
		D S Threads II B Heap B Allocation Tracker	le Explorer			-
		ID. The Design of the later being	in coporti			
android.process.media 352	Start Method Prof	1 534 native 235 88 main				
com.android.mms 366	8650	*2 535 vmwalt 5 167 HeapWorker				
com android smail 388	8651	*3 536 smealt 0 0 GC *4 537 smealt 0 0 Signal Catcher				
com.android.camera 428	8652	*5 538 running 443 822 JOMP				
com.android.gallery 459	8653	16 539 vmwait 0 0 Compiler				
tourse.sample 534	3 8545 / 870K	7 540 native 0 0 Binder Thread #1				
com.android.browser 926	8655	8 541 native 0 0 Binder Thread #2				
com.android.developmen 1007	8610	9 542 native 0 0 Binder Thread #3				
com.android.term 1212	8654	T				
Emulator Control 12	6	° 🗆				
Telephony Status		0				
Voice: home Soc	ed: full	(Refresh) Thu May 05 17:06:20 EDT 2011				
		Kerresh Thu May 05 17:06:20 EDT 2011				
Data: home 🚺 Las	ency: None	Class	Method	File	Line Native	
		android.os.MessageQueue	nativePollOnce	MessageQueue.java	-2 true	
Telephony Actions		android.os.MessageQueue	next	MessageQueue.Java	119 false	
Incoming number		android.os.Looper android.app.ActivityThread	loop	Looper.java ActivityThread.java	3683 false	
· Voice		java lang reflect Method	invokeNative	Method.iava	-2 true	
() SMS		Java lang reflect.Method	invoke	Method Java	507 faise	
Message:		com.android.internal.os.ZygoteinitSMethodAndArgsCalle		Zygotehit.java	839 false	
wessage.		com.android.internal.os.Zygoteinit dalvik.system.NativeStart	main	Zygotelnit.java NativeStart.java	597 false +2 true	
			man	Nativeskart.java	- 2 Inte	
a constant	1.12				0000+	
Console StogCat 23						M
		Log				
Time pid tag	Message					
05-05 17:07:55 3 534 dalvikus 05-05 17:07:55 3 534 dalvikus	threadidal: still s	urpended after undo (scal dos1) freed 23E, 53E free 2560E/5379E, enternal 716E/1630E, name				n
05-05 17.00:29. I 75 Activitellar	sour Displayed course. sa	sple/course emandes HelloRadroid +53s08ms				
05-05 17 09:55 3 75 SateClient 05-05 17 09:11 3 594 dalvikus	resuest time failed	isva met SocketEsception: Address family not supported b	retocol			
05-05 17 99:11 I 534 dalvikum	TRACE STRATED 110	151 019203				
05-05 17:09:36.2 534 dalvikve 05-05 17:09:37 I 534 dalvikve	TRACE STOPPED with	count now 0				
05-05 17:09:07 # 554 dalvikue	distriction of determiner	is active, wethod-tracing results will be showed				
05-05 17 29:57 3 534 dalvikve	*** active profiler					0
05-05 17 10 27 3 534 dalvikum	*** active profiler	count now 0				<u>ب</u>
05-05 17 10 27 1 534 dalvibwe (5-05 17 10 27 5 534 dalvibwe	TRACE STOPPED with WAREHOUS a debugger	ing 493 records is active, method-tracing receipt will be showed				4
Filter:						_
						-



See en.wikipedia.org/wiki/Heisenbug

 Some concurrency complexities can be fixed by applying Java built-in monitor object mechanisms



 There are also helpful techniques for debugging concurrent software



Multithreaded Debugging Techniques

By Shameem Akhter and Jason Roberts, April 23, 2007

Post a Comment

Debugging multithreaded applications can be a challenging task.

Debugging multithreaded applications can be a challenging task. The increased complexity of multithreaded programs results in a large number of possible states that the program may be in at any given time. Determining the state of the program at the time of failure can be difficult; understanding why a particular state is troublesome can be even more difficult. Multithreaded programs often fail in unexpected ways, and often in a nondeterministic fashion. Bugs may manifest themselves in a sporadic fashion, frustrating developers who are accustomed to troubleshooting issues that are consistently reproducible and predictable. Finally, multithreaded applications can fail in a drastic fashion-deadlocks cause an application or worse yet, the entire system, to hang. Users tend to find these types of failures to be unacceptable.

General Debug Techniques

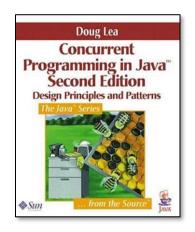
Regardless of which library or platform that you are developing on, several general principles can be applied to debugging multithreaded software applications.

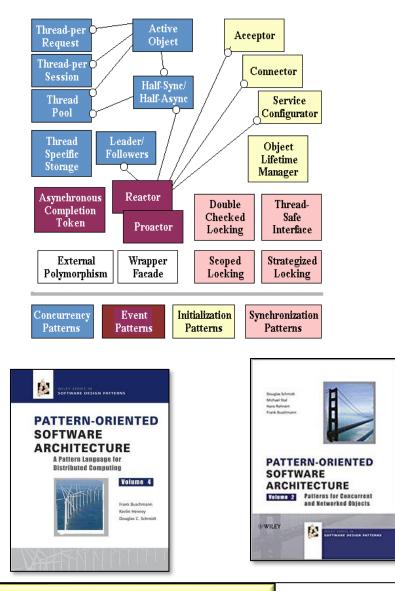
The first technique for eliminating bugs in multithreaded code is to avoid introducing the bug in the first place. Many software defects can be prevented by using proper software development practices. The later a problem is found in the product development lifecycle, the more expensive it is to fix. Given the complexity of multithreaded programs, it is critical that multithreaded applications are properly designed up front.

How often have you, as a software developer, experienced the following situation? Someone on the team that you're working on gets a great idea for a new product or feature. A quick prototype that illustrates the idea is implemented and a quick demo, using a trivial use-case, is presented to management. Management loves the idea and immediately informs sales and marketing of the new product or feature. Marketing then informs the customer of the feature, and in order to make a sale, promises the customer the feature in the next release. Meanwhile, the engineering team, whose original intent of presenting the idea was to get resources to properly implement the product or feature sometime in the future, is now faced with the task of delivering on a customer commitment immediately. As a result of time constraints, it is often the case that the only option is to take the prototype, and try to turn it into production code.

See www.drdobbs.com/cpp/multithreaded-debugging-techniques/199200938

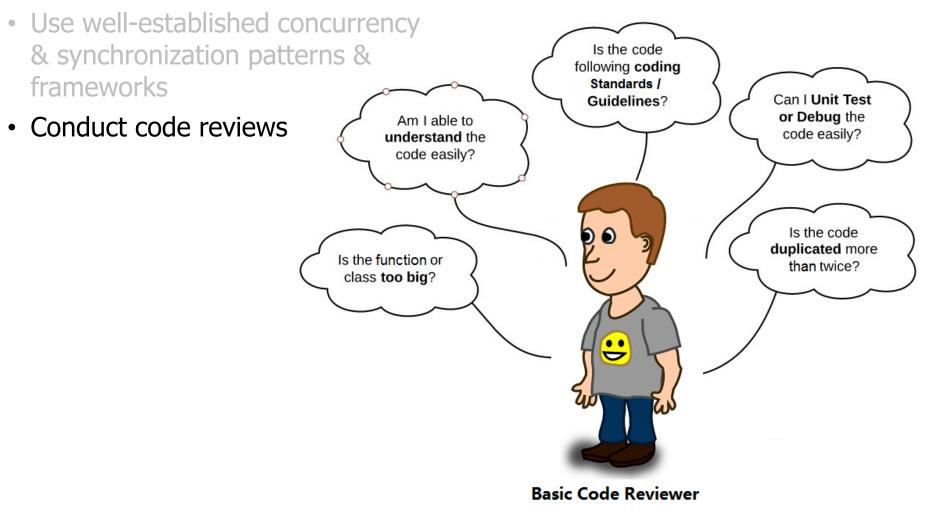
- There are also helpful techniques for debugging concurrent software, e.g.
 - Use well-established concurrency & synchronization patterns & frameworks





See <u>en.wikipedia.org/wiki/Concurrency_pattern</u> & <u>www.dre.Vanderbilt.edu/~schmidt/frameworks.html</u>

• There are also helpful techniques for debugging concurrent software, e.g.



See <u>en.wikipedia.org/wiki/Code_review</u>

- There are also helpful techniques for debugging concurrent software, e.g.
 - Use well-established concurrency & synchronization patterns & frameworks
 - Conduct code reviews
 - Apply automated analysis tools

Static Analysis Tools for Concurrency

- FindBugs works on Java. In the list of bugs detected all of the "Multithreaded correctness" bugs are relevant to concurrency. Command-line interface or eclipse plugin (eclipse plugin update site:http://findbugs.cs.umd.edu/eclipse/)
- Lint a UNIX tool for C
- JLint a Java version of Lint that is available as stand alone or eclipse plugin (eclipse plugin update site:http://www.jutils.com/eclipse-update)
- Parasoft JTest commercial tool that combines static analysis and testing. Has capability to check for thread safety in multithreaded Java programs.
- Coverity Static Analysis and Static Analysis Custom Checkers commercial tool that can be used to create custom static analyzers to find concurrency bugs in C/C++ programs.
- GrammaTech's CodeSonar commercial tool that can detect a special case race condition and locking issues in C/C++ (see datasheet for list of all bugs detected).
- Chord static and dynamic analysis tool for Java (listed above as well).
- JSure for Concurrency a commercial tool from SureLogic that is currently available in early release.
- ESC/Java 2 can detect race conditions and deadlocks requires annotation (more...)
- Relay static race detection
- RacerX uses flow-sensitive static analysis tool for detection race conditions and deadlocks in C [paper] [slides]
- SyncChecker a tool developed by F. Otto and T. Moschny for finding race conditions and deadlocks in Java. Reduce false positives by combining static analysis with points-to and may-happen-in-parallel (MHP) information.
- Warlock race detection tool for C requires annotation.

See www.sqrlab.ca/blog/2012/03/02/static-analysis-tools-for-concurrency

- There are also helpful techniques for debugging concurrent software, e.g.
 - Use well-established concurrency & synchronization patterns & frameworks
 - Conduct code reviews
 - Apply automated analysis tools
 - Instrument code with logging & tracing statements

```
C:\WINDOWS\system32\cmd.exe
                                                                          - 🗆 X
     main
     -> base_class::base_class
     counter=[0]
     param=[setec astronomy]
    <-- base_class::base_class in Øs
     -> class_a
     -- class_a in 0.015s
-> base_class∷foobar
        -> base_class::foo
      <-- base_class::foo in Øs
       -> class_a::bar
          -> class_a::some_helper
           x=[1]
          y=[10]
          z=[a]
         <--_class_a::some_helper in Os
     <-- class_a::bar in 0.016s
counter=[109]</pre>
    <-- base_class::foobar in 0.016s
    --> base_class::base_class
     counter=[0]
     param=[default b param]
        base_class::base_class in 0s
     -> class_b
     -- class_b in Øs
-> base_class∷foobar
        > base_class::foo
      <-- base_class::foo in Os
       --> class_b::bar
        will fail=[0]
         --> base_class::bar
        <-- base_class::bar in 0.015s</pre>
      <-- class_b::bar in 0.015s
     counter = [\overline{2}]
   <-- base_class::foobar in 0.015s
all methods called, compiling results
collection(integer_list, 2 items)</pre>
       [0] = [109]
       [1] = [2]
   3
   --> base_class::base_class
   counter=[0]
param=[default b param]
<-- base_class::base_class in 0s</pre>
   --> class_b
   <-- class_b in 0.016s
    --> base_class::foobar
      --> base_class::foo
      <-- base_class::foo in Øs
      --> class_b::bar
        will_fail=[1]
     throwing an exception as I was configured to fail <-- class_b::bar in 0s
   <-- base_class::foobar in Øs
bound to happend
 <-- main in 0.093s
Press any key to continue . . . _
```

See www.dre.vanderbilt.edu/~schmidt/PDF/DSIS_Chapter_Waddington.pdf

End of Evaluating the Java Monitor Object Motivating Example