Session: Concurrency and Parallelism in Java

....................................................…

1. Hardware und Compiler:

Stalking the Lost Write: Memory Visibility in Concurrent Java Jeff Berkowitz, New Relic

December 2013

(im gleichen Verzeichnis).

Schauen Sie sich die sog. „Executions“ der beiden Threads nochmal an. Erkennen Sie, dass es nur die zufällige Ablaufreihenfolge der Threads ist, die die verschiedenen Ergebnisse erzeugt? Das sind die berüchtigten „race conditions“…

Tun Sie mir den Gefallen und lassen Sie das Programm Mycounter in den se2\_examples selber ein paar Mal ablaufen. Tritt die Race Condition auf? Wenn Sie den Lost Update noch nicht verstehen: Bitte einfach nochmal fragen!

Tipp: Wir haben ja gesehen, dass i++ von zwei Threads gleichzeitig auseführt increments verliert (lost update). Wir haben eine Lösung gesehen in MyCounter: das Ganze in einer Methode verpacken und die Methode „synchronized“ machen. Also die „Ampellösung“ mit locks dahinter. Das ist teuer und macht unseren Gewinn durch mehr Threads kaputt.

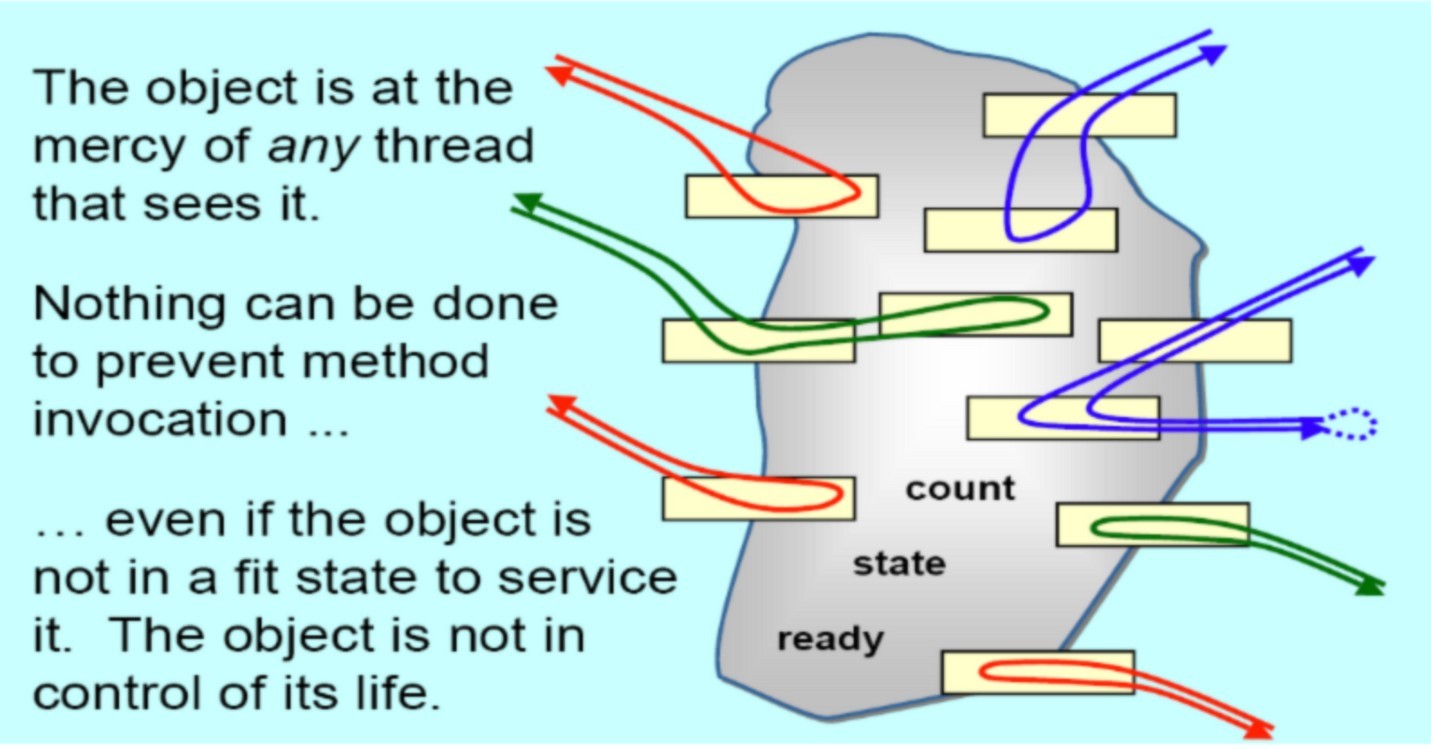
Hm, könnte man i++ ATOMAR updaten ohne Locks??

Suchen Sie mal nach „AtomicInteger“ im concurrent package von Java!

1. Arbeitet Ihr Gehirn concurrent oder parallel (:-)?

Parallel, da man mehrere Dinge gleichzeitig tun, denken kann

1. Multi-threading hell: helfen „private, protected, public“ dagegen? Nein



Ist Ihnen klar warum die OO-Kapselung durch „private“ Felder nicht hilft bei Multithreading?

Es können mehrere Threads die Variablen durch setter Methoden immer noch verändern

1. Muss der schreibende Zugriff von mehreren Threads auf folgende Datentypen synchronisiert werden?

* statische Variablen
* Felder/Attribute in Objekten
* Stackvariablen
* Heapvariablen (mit new() allokierte..)

Statische Variablen: Ja

Felder in Objekten: Ja, solange das Objekt von mehreren Threads verwendet wird.

Stack-Variablen: Nein, da Stack-Variablen nicht gemeinsam genutzt werden.

Heap-Variablen: Ja

1. Machen Sie die Klasse threadsafe (sicher bei gleichzeitigem Aufruf). Gibt‘s Probleme? Schauen Sie ganz besonders genau auf die letzte Methode getList() hin.

class Foo {

private ArrayList<Object> aL = new ArrayList<>(); public int elements;

private String firstName; private String lastName;

public Foo () {};

public void synchronized setfirstName(String fname) { firstName = fname;

elements++;

}

public setlastName(String lname) { lastName = lname; elements++;

}

public synchronized List<Object> getList () { return Collections.unmodifiableList(aL);

}

}

1. kill\_thread(thread\_id) wurde verboten: Wieso ist das Töten eines Threads problematisch? Wie geht es richtig?

Es können Daten verloren, bzw. kaputt gehen, wenn während des Thread-Ablaufs er geschlossen wird

1. Sie lassen eine Applikation auf einer Single-Core Maschine laufen und anschliessend auf einer Multi- Core Maschine. Wo läuft sie schneller?

Wenn Applikation mit mehreren Threads arbeitet, dann läuft es auf Multi-Core schneller, bei wenig Daten eher bei Single-Core

1. Was verursacht Non-determinismus bei multithreaded Programmen? (Sie wollen in einem Thread auf ein Konto 100 Euro einzahlen und in einem zweiten Thread den Kontostand verzehnfachen)

Lost update, da beide gleichzeitig verändern

1. Welches Problem könnte auftreten wenn ein Thread produce() und einer consume() aufruft? Wie sähen Lösungsmöglichkeiten aus? Wenn es mehr als einen Producer und/oder Consumer gibt? Wo könnte/sollte

„volatile“ hin? Wo eventuell „synchronized“?

1 public class MyQueue { 2

1. private Object store;
2. int flag = false; // empty

Synchronized

1. public void produce(Object in) {
2. while (flag == true) ; //full
3. store = in;

flag = true; //full

9 } Synchronized

1. public Object consume() {

Object ret;

1. while (flag == false) ; //empty
2. ret = store;
3. flag = false; //empty
4. return ret; 15 }

16 }

1. Fun with Threads:
   1. Was passiert mit dem Programm?
   2. Was kann auf der Konsole stehen?

**public class** C1 {

**public synchronized void** doX (C2 c2) { c2.doX();

}

**public synchronized void** doY () { System.***out***.println(**"Doing Y"**);

}

**public static void** main (String[] args) { C1 c1 = **new** C1();

C2 c2 = **new** C2();

Thread t1 = **new** Thread(() -> { **while** (**true**) c1.doX(c2);}); Thread t2 = **new** Thread(() -> { **while** (**true**) c2.doY(c1);}); t1.start();

t2.start();

}

};

**public class** C2 {

**public synchronized void** doX () { System.***out***.println(**"Doing X"**);

}

**public synchronized void** doY ( C1 c1) { c1.doY();

}

* Unendlich oft „Doing X“ oder „Doing Y“ in der Konsole
* Deadlock