Session: Concurrency and Parallelism in Java
.....…

0. 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!

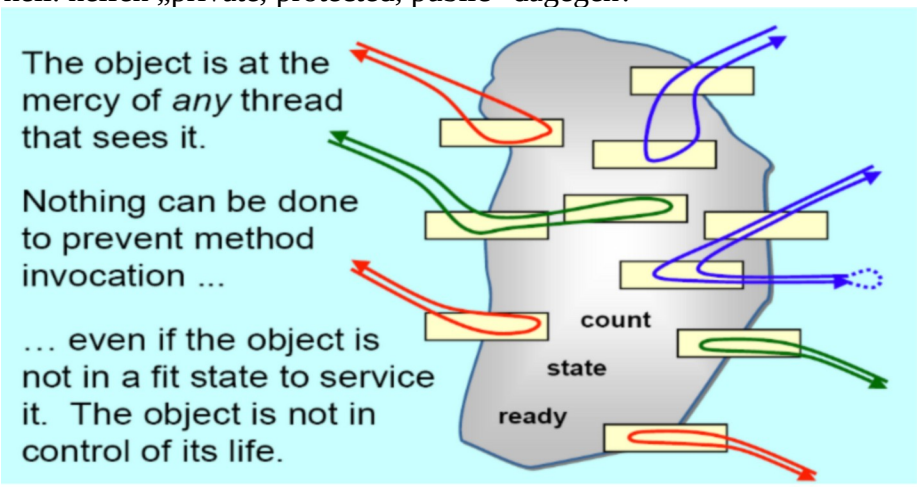
Tipp: Wir haben ja gesehen, dass i++ von zwei Threads gleichzeitig auseführt increments verliert (lost update). Wir haben eine Lösung gesehen in MyCounter: das Ganze in einer Methode verpacken und die Methode „synchronized“ machen. Also die „Ampellösung“ mit locks dahinter. Das ist teuer und macht unseren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

Physisch parallel, Man kann denken und schreiben gleichzeitig(Gehirn steuert auch Bewegung)

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



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

Nein hilft nicht da es 2 Threads nicht daran hindert in der gleichen Klasse zu arbeiten
Man braucht synchronized

3. Muss der schreibende Zugriff von mehreren Threads auf folgende Datentypen synchronisiert werden?
- statische Variablen ja
- Felder/Attribute in Objekten ja

- Stackvariablen nein
- Heapvariablen (mit new() allokierte..) ja

4. 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 aL = new ArrayList();
private  ~~public~~ int elements;
        private String firstName;
        private String lastName;

        public Foo () {};

        public void synchronized setfirstName(String fname) {
                firstName = fname;
                elements++;
        }                                   weil beides auf elements zugreift
        synchronized
        public setlastName(String lname) {
                lastName = lname;
                elements++;
        }
        public ~~synchronized~~ ArrayList getList () {      wird nicht von den andern Methoden beeinflusst
        return aL;   .clone
    }
}
```

5. kill_thread(thread_id) wurde verboten: Wieso ist das Töten eines Threads problematisch? Wie geht es richtig?

Kann methode mittendrin stoppen und somit das Programm breaken.
Man kann es über yield(),notify(), wait(), .interrupt()
oder über einen Executer machen

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

kommt auf die Menge der Threads und den Code an. Auch auf einer Singlecore Maschine können mehrere Threads "parallel" durch concurrency laufen.
Manchmal ist weiter geben schneller als auf Threads zu warten

7. 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)

Kann falsch herum ablaufen, also erst wird verzehnfacht und dann addiert
Weil CPU nicht auf Reihenfolge achtet wenn Zusammenhang nicht sofort ersichtlich(menschliche Logik und zusammenhänge != compilerlogik)

8. Welches Problem könnte auftreten wenn ein Thread produce() und einer consume() aufruft? Wie sähen Lösungsmöglichkeiten aus?

```
1  public class MyQueue {
2
3      private Object store;
4      int flag = false; // empty
5      public  void produce(Object in) {
6              while (flag == true)   ;  //full
7              store = in;
               flag = true; //full
9      }
10     public  Object consume() {
               Object ret;
11             while (flag == false)    ; //empty
12             ret = store;
13             flag = false;   //empty
14             return ret;
15     }
16 }
```

boolean

flag ist nicht volatile(aus Sicht des compilers ändert sich flag nie) -> volatile

Probleme gibt es wenn 2 Threads die gleiche Methode aufrufen,
dann müsste man es mit einem synchronized lösen

9. Fun with Threads:

1. Was passiert mit dem Programm? deadlock das Programm beendet sich nicht weil c1 auf c2, was blockiert ist zugreifen willl
2. Was kann auf der Konsole stehen? wenn thread 1 zuerst aufgerufen wurde doing x und umgekehrt

```
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();
}
```