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!

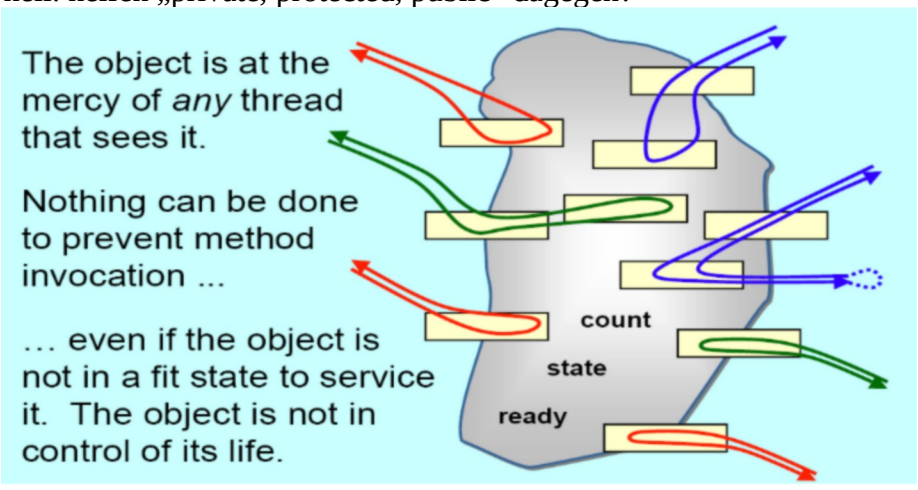
Tipp: Wir haben ja gesehen, dass i++ von zwei Threads gleichzeitig auseführt increments verliert (lost update). Wir haben eine Lösung gesehen in MyCounter: das Ganze in einer Methode verpacken und die Methode „synchronized“ machen. Also die „Ampellösung“ mit locks dahinter. Das ist teuer und macht unseren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

mixture of both

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



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

even private fields can be modified by multiple threads

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

- Stackvariablen yes
- Heapvariablen (mit new() allokierte..) no

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();
        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 ArrayList getList () {
        return aL;
    }
}
```

5. kill_thread(thread_id) wurde verboten: Wieso ist das Töten eines Threads problematisch? Wie geht es richtig? When killing a thread, you never know at what point it is stopped exactly. → Could cause unexpected behaviour e.g. when setting variables.

Better: interrupt thread

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

If the application can run on multiple cores, that way would be faster. Generally: Depends on application, processors, ...

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)

The order could possible be changed so that the number is multiplied by 10 first and then increased by 100 which would lead to a completely different result.

8. 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
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 }
```

9. 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");
```

→ create objects c1 and c2

} threads call 2 diff. methods

```
 }
 public synchronized void doY ( C1 c1) {
  c1.doY();
}
```

because of the synchronized, only the first Thread is actually run its "while (true)" - loop

→ Console output is "doing X", keeps running infinitely.