|  |  |
| --- | --- |
|  | **2015** |
|  | The Company  Martien Huijsmans |

|  |
| --- |
| **[Java-Threading]** |
| Getting threading right |

Inhoud

[Introduction 1](#_Toc432932919)

[Scope 1](#_Toc432932920)

[References 1](#_Toc432932921)

# Introduction

This document discuss java threading and how to avoid “data race”

This document is written using the Java Language specification and examples on stackoverflow. See references for used links.

# Scope

Java >= 1.5. In Java SE 5 Java Memory Management Model has been updated.

# Java Specification

This chapter contains quotes from the Java SE 8 specification.

§ 17.4.3 Programs and Program Order

Among all the inter-thread actions performed by each thread t, the program order of t is a total order that reflects the order in which these actions would be performed according to the intra-thread semantics of t.

*§ 17.4.4  
A write to a volatile variable v (§8.3.1.4) synchronizes-with all subsequent reads of v by any thread (where "subsequent" is defined according to the synchronization order)*

§ *17.4.5 Happens-before Order*

*Two actions can be ordered by a happens-before relationship. If one action happens-before another, then* ***the first is visible to and ordered before the second****.  
If we have two actions x and y, we write hb(x, y) to indicate that x happens-before y.*

*• If x and y are actions of the same thread and x comes before y in program order, then hb(x, y).*

*• There is a happens-before edge from the end of a constructor of an object to the start of a finalizer (§12.6) for that object.*

*• If an action x synchronizes-with a following action y, then we also have hb(x, y).*

*• If hb(x, y) and hb(y, z), then hb(x, z).*

§ *17.4.5 Happens-before Order*It follows from the above definitions that:  
  
• An unlock on a monitor happens-before every subsequent lock on that monitor.

• A write to a volatile field (§8.3.1.4) happens-before every subsequent read of that field.

• A call to start() on a thread happens-before any actions in the started thread.

• All actions in a thread happen-before any other thread successfully returns from a join() on that thread.

• The default initialization of any object happens-before any other actions (other than default-writes) of a program

# Volatile and shared fields

Example:

class Test {

private volatile boolean serviceReady = false;

private Service service;

private Translator translator;

public void setService(Service service, Translator translator;) {

this.service = service;

this.translator = translator;

this.serviceReady = true;  
 }

public void doWork() {

if ( serviceReady ) {

service.doWork(translator);

}

}

}

Question: when thread T1 calls setService(..) and thread T2 calls doWork() will thread T2 see correct value of field service??  
Answer: yes  
Rationale (using quotes from JLS)

1. "If x and y are actions of the same thread and x comes before y in program order, then hb(x, y)"
2. "A write to a volatile field (§8.3.1.4) happens-before every subsequent read of that field."
3. "If hb(x, y) and hb(y, z), then hb(x, z)" (transitivity of happens-before)

Question: Is this thread safe?  
A: class is not thread safe.   
If thread T1 and T2 calls setService(..) the result can be that service of T1 and translator of T2 is stored.  
T2: store service  
T1: store service  
T1: store translator  
T2: store translator

# Synchronized (block) and shared variables

Example:

class Test {

private final Object lock = new Object();

private Service service;

private Translator translator;

public void setService(Service service, Translator translator) {

synchronized(lock) {

this.service = service;

this.translator = translator;

this.translator = translator;

}

}

public void doWork() {

synchronized(lock) {

if (service != null) {

service.doWork(translator);

}

}

}

# Java concurrent package & memory consistency

*Chapter 17 of the Java Language Specification defines the happens-before relation on memory operations such as reads and writes of shared variables. The results of a write by one thread are guaranteed to be visible to a read by another thread only if the write operation happens-before the read operation. The synchronized and volatile constructs, as well as the Thread.start() and Thread.join() methods, can form happens-before relationships. The methods of all classes in java.util.concurrent and its subpackages extend the guarantees to higher-level synchronization*: <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html#MemoryVisibility>

As example, the CountCountLatch (<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CountDownLatch.html>) states: Memory consistency effects: Until the count reaches zero, actions in a thread prior to calling countDown() [*happen-before*](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html#MemoryVisibility) actions following a successful return from a corresponding await()in another thread.  
Note the conditional before-after (“successful return”, i.e. “before-after” only applies when await(..) returns true).

# Cache line: 64 bytes

# References

|  |  |
| --- | --- |
| Ref |  |
| Java SE spec | http://docs.oracle.com/javase/specs/ |
| Volatile and shared fields | <http://stackoverflow.com/questions/1351168/volatile-semantic-with-respect-to-other-fields?lq=1> |
|  | <http://stackoverflow.com/questions/30246007/java-memory-model-volatile-variables-and-happens-before?lq=1> |