Java “Happens-Before” Relationships: Examples

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
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 “happens-before” relationships mean in Java
- Recognize how Java Thread methods support “happens-before” relationships
Learning Objectives in this Part of the Lesson

- Understand what “happens-before” relationships mean in Java
- Recognize how Java Thread methods support “happens-before” relationships
- Know how Java collections support “happens-before” relationships

ConcurrentHashMap
Java Thread “Happens-Before” Relationships
Java Thread “Happens-Before” Relationships

- Methods in the Java Thread class establish “happen-before” relationships

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html
Java Thread “Happens-Before” Relationships

- Methods in the Java Thread class establish “happen-before” relationships
- Starting a thread “happens-before” the run() hook method of the thread is called

```java
Thread threadB = new Thread( );
............
............
threadB.start( );
............
```

```java
............
public void run( ){
    statement 1;
    ........
    ........
}
............
```

threadB.start() happened before all statements in run
Methods in the Java Thread class establish “happen-before” relationships

Starting a thread “happens-before” the run() hook method of the thread is called

```java
Thread t1 =
    new Thread(() ->
        System.out.println
            ("hello world");

t1.start();

...
```
Java Thread “Happens-Before” Relationships

- Methods in the Java Thread class establish “happen-before” relationships
- Starting a thread “happens-before” the run() hook method of the thread is called

```java
Thread t1 =
    new Thread(() ->
        System.out.println("hello world"));

t1.start();

... Create & start a new thread
```
Java Thread “Happens-Before” Relationships

- Methods in the Java Thread class establish “happen-before” relationships
- Starting a thread “happens-before” the run() hook method of the thread is called

```java
Thread t1 =
    new Thread(() ->
        System.out.println("hello world"));

t1.start();
```

This lambda expression plays the role of the run() hook method!
Java Thread “Happens-Before” Relationships

- Methods in the Java Thread class establish “happen-before” relationships
- Starting a thread “happens-before” the run() hook method of the thread is called

```java
Thread t1 =
    new Thread(() ->
        System.out.println
            ("hello world"));

    t1.start();

...  
```

The state of thread t1 is consistent & visible before run() begins to execute
Java Thread “Happens-Before” Relationships

- Methods in the Java Thread class establish “happen-before” relationships
  - Starting a thread “happens-before” the run() hook method of the thread is called
  - The termination of a thread “happens-before” a join() with the terminated thread
• Methods in the Java Thread class establish “happen-before” relationships
  • Starting a thread “happens-before” the run() hook method of the thread is called
  • The termination of a thread “happens-before” a join() with the terminated thread

```java
Thread t1 =
    new Thread(() ->
        System.out.println("hello world");
    t1.start();

...;

t1.join();
```
Java Thread “Happens-Before” Relationships

- Methods in the Java Thread class establish “happen-before” relationships
  - Starting a thread “happens-before” the run() hook method of the thread is called
  - The termination of a thread “happens-before” a join() with the terminated thread

```java
Thread t1 = new Thread(() -> 
    System.out.println("hello world"));
t1.start();
...
t1.join();
```

Thread t1 terminates after its lambda expression run() processing completes
Methods in the Java Thread class establish “happen-before” relationships

Starting a thread “happens-before” the run() hook method of the thread is called

The termination of a thread “happens-before” a join() with the terminated thread

```
Thread t1 =
    new Thread(() ->
        System.out.println("hello world"));

    t1.start();

...

t1.join();
```

The thread waiting on join() only resumes it’s processing after the thread t1 terminates
Java Collections “Happens-Before” Relationships
Java Collections “Happens-Before” Relationships

- Methods in java.util.concurrent package classes also establish “happen-before” relationships

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html#MemoryVisibility
Java Collections “Happens-Before” Relationships

- Methods in java.util.concurrent package classes also establish “happens-before” relationships
- The release of a monitor lock “happens-before” every subsequent acquire on the same lock

Methods in java.util.concurrent package classes also establish “happen-before” relationships

The release of a monitor lock “happens-before” every subsequent acquire on the same lock

```java
class ArrayBlockingQueue<E> {
    public void put(E e) {
        final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
        try {
        } finally {
            lock.unlock();
        }
    }
}

class ArrayBlockingQueue<E> {
    public E take() {
        final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
        try {
        } finally {
            lock.unlock();
        }
    }
}
```
Consider the put() & take() methods in `ArrayBlockingQueue`
Java Collections “Happens-Before” Relationships

- Methods in java.util.concurrent package classes also establish “happen-before” relationships
- The release of a monitor lock “happens-before” every subsequent acquire on the same lock

```java
class ArrayBlockingQueue<E> { ...
    public void put(E e) { ...
        final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
        try { ... } finally {
            lock.unlock();
        }
    }
}
```

```java
class ArrayBlockingQueue<E> { ...
    public E take() { final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
        try { ... } finally {
            lock.unlock();
        }
    }
}
```

*Actions prior to "releasing" the ReentrantLock must happen-before actions subsequent to a successful "acquiring" of this lock*

See earlier lessons on “Java ReentrantLock” & “Java ConditionObject”
Java Collections “Happens-Before” Relationships

• Methods in java.util.concurrent package classes also establish “happen-before” relationships
• The release of a monitor lock “happens-before” every subsequent acquire on the same lock
• Actions in a thread prior to placing an object into any concurrent collection “happen-before” actions subsequent to the access or removal of that element from the collection in another thread

```java
ConcurrentMap concurrentMap = new ConcurrentHashMap();

// Thread t1
c concurrentMap.put("key", "value");

// Thread t2
Object value = concurrentMap.get("key");
```
Java Collections “Happens-Before” Relationships

- Methods in java.util.concurrent package classes also establish “happen-before” relationships
- The release of a monitor lock “happens-before” every subsequent acquire on the same lock
- Actions in a thread prior to placing an object into any concurrent collection “happen-before” actions subsequent to the access or removal of that element from the collection in another thread

```java
ConcurrentMap concurrentMap = new ConcurrentHashMap();

// Thread t1
concurrentMap.put("key", "value");

// Thread t2
Object value = concurrentMap.get("key");
```

Consider a ConcurrentHashMap that supports concurrent retrievals & high expected concurrency for updates

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html
Java Collections “Happens-Before” Relationships

- Methods in java.util.concurrent package classes also establish “happen-before” relationships
  - The release of a monitor lock “happens-before” every subsequent acquire on the same lock
  - Actions in a thread prior to placing an object into any concurrent collection “happen-before” actions subsequent to the access or removal of that element from the collection in another thread

```java
ConcurrentMap concurrentMap = new ConcurrentHashMap();

// Thread t1
concurrentMap.put("key", "value");

// Thread t2
Object value = concurrentMap.get("key");
```

Placing a “key/value” element into a ConcurrentHashMap must happen-before accessing or removing this element from the map
Java’s class libraries are responsible for ensuring these “happens-before” relationships are preserved.
Java Collections “Happens-Before” Relationships

- Java’s class libraries are responsible for ensuring these “happens-before” relationships are preserved.

You don’t need to understand all the nitty-gritty details of Java’s memory model – you just need to understand how to use synchronizers properly!
End of “Happens-Before” Relationships: Examples