Java ReentrantLock

(Part 4)

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 how the concept of mutual exclusion in concurrent programs
• Recognize how Java ReentrantLock provides mutual exclusion to concurrent programs
• Know the key methods defined by the Java ReentrantLock class
• Learn how to use ReentrantLock in Java programs

Class ArrayBlockingQueue\(<E>\>

java.lang.Object
  java.util.AbstractCollection\(<E>\>
    java.util.AbstractQueue\(<E>\>
      java.util.concurrent.ArrayBlockingQueue\(<E>\>

Type Parameters:
E - the type of elements held in this collection

All Implemented Interfaces:
Serializable, Iterable\(<E>\), Collection\(<E>\), BlockingQueue\(<E>\), Queue\(<E>\>

public class ArrayBlockingQueue\(<E>\>
  extends AbstractQueue\(<E>\>
  implements BlockingQueue\(<E>\), Serializable

A bounded blocking queue backed by an array. This queue orders elements FIFO (first-in-first-out). The head of the queue is that element that has been on the queue the longest time. The tail of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue.

This is a classic "bounded buffer", in which a fixed-sized array holds elements inserted by producers and extracted by consumers. Once created, the capacity cannot be changed. Attempts to put an element into a full queue will result in the operation blocking; attempts to take an element from an empty queue will similarly block.

This class supports an optional fairness policy for ordering waiting producer and consumer threads. By default, this ordering is not guaranteed. However, a queue constructed with fairness set to true grants threads access in FIFO order. Fairness generally decreases throughput but reduces variability and avoids starvation.
Using Reentrant Lock in Java
Using ReentrantLock in Java

- ArrayBlockingQueue is a blocking bounded FIFO queue

```
public class ArrayBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>,
    java.io.Serializable {

Class ArrayBlockingQueue<E>

java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractQueue<E>
            java.util.concurrent.ArrayBlockingQueue<E>

Type Parameters:
    E - the type of elements held in this collection

All Implemented Interfaces:
    Serializable, Iterable<E>, Collection<E>, BlockingQueue<E>, Queue<E>

public class ArrayBlockingQueue<E>
extends AbstractQueue<E>
implements BlockingQueue<E>, Serializable

A bounded blocking queue backed by an array. This queue orders elements FIFO (first-in-first-out). The head of the queue is that element that has been on the queue the longest time. The tail of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue.
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html
Using ReentrantLock in Java

- ArrayBlockingQueue is a blocking bounded FIFO queue

```java
public class ArrayBlockingQueue<E> extends AbstractQueue<E>
    implements BlockingQueue<E>, java.io.Serializable {
```

See [docs.oracle.com/javase/8/docs/api/java/util/AbstractQueue.html](docs.oracle.com/javase/8/docs/api/java/util/AbstractQueue.html)
Using ReentrantLock in Java

• ArrayBlockingQueue is a blocking bounded FIFO queue

```
public class ArrayBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>,
    java.io.Serializable {
 ...
```

---

**Interface BlockingQueue<E>**

Type Parameters:

E - the type of elements held in this collection

All Superinterfaces:

Collection<E>, Iterable<E>, Queue<E>

All Known Subinterfaces:

BlockingDeque<E>, TransferQueue<E>

All Known Implementing Classes:

ArrayBlockingQueue, DelayQueue, LinkedBlockingDeque, LinkedBlockingQueue, LinkedTransferQueue, PriorityBlockingQueue, SynchronousQueue

```
public interface BlockingQueue<E>
extends Queue<E>
```

A Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html)
Using ReentrantLock in Java

- ArrayBlockingQueue is a blocking bounded FIFO queue

```java
public class ArrayBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>,
    java.io.Serializable {
    ...
```

We’ll consider both the interface & implementation of ArrayBlockingQueue
• ArrayBlockingQueue is a blocking bounded FIFO queue

```java
public class ArrayBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>,
    java.io.Serializable {

    // Main lock guarding all access
    final ReentrantLock lock;

    // The queued items
    final Object[] items;

    // items indices for next take
    // or put calls
    int takeIndex;
    int putIndex;

    // Number of elements in the queue
    int count;
```

See [www.dre.vanderbilt.edu/~schmidt/C++2java.html#concurrency](http://www.dre.vanderbilt.edu/~schmidt/C++2java.html#concurrency)
Using ReentrantLock in Java

- ArrayBlockingQueue is a blocking bounded FIFO queue

```java
public class ArrayBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>,
    java.io.Serializable {

    ...

    // Main lock guarding all access
    final ReentrantLock lock;
    ...

    // The queued items
    final Object[] items;

    // items indices for next take
    // or put calls
    int takeIndex;
    int putIndex;

    // Number of elements in the queue
    int count;

    Object state that’s being protected by the lock
```
ArrayBlockingQueue is a blocking bounded FIFO queue

public class ArrayBlockingQueue<E> extends AbstractQueue<E> implements BlockingQueue<E>, java.io.Serializable {

    final ReentrantLock lock;

    final Object[] items;

    int takeIndex;
    int putIndex;

    int count;

    // Main lock guarding all access
    final ReentrantLock lock;

    // The queued items
    final Object[] items;

    // items indices for next take
    // or put calls
    int takeIndex;
    int putIndex;

    // Number of elements in the queue
    int count;

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html
• ArrayBlockingQueue is a blocking bounded FIFO queue

```java
ArrayBlockingQueue<String> q = new ArrayBlockingQueue<String>(10);
...
// Called by thread T1
String s = q.take();
...
```

Thread $T_1$ acquires the lock & enters the critical section
Using ReentrantLock in Java

- ArrayBlockingQueue is a blocking bounded FIFO queue

```java
class ArrayBlockingQueue<E> extends AbstractQueue<E>
    implements BlockingQueue<E>, java.io.Serializable {
    ...
    public E take() ... {
        final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
        ...
    }

    The lock's hold count is incremented by 1
```
Using ReentrantLock in Java

- ArrayBlockingQueue is a blocking bounded FIFO queue

```java
// Called by thread T2
String s = q.take();
```

ArrayBlockingQueue

- unlocked (holdCount = 0)
- locked (holdCount = 1)

Critical Section

A call to take() from thread T₂ will block until thread T₁ is finished
public class ArrayBlockingQueue\<E> extends AbstractQueue\<E> implements BlockingQueue\<E>, java.io.Serializable {

    public E take() ... {
        final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
        try {
            ...  
        } finally {
            lock.unlock();
        }
    } ...

    When thread $T_1$ finishes in take() it unlocks the lock

    unlocked (holdCount = 0)

    locked (holdCount = 1)

    Critical Section

    unlock()
Using ReentrantLock in Java

- ArrayBlockingQueue is a blocking bounded FIFO queue

```java
public class ArrayBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>,
    java.io.Serializable {

    public E take() ... {
        final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
        try { ... 
            } finally {
                lock.unlock();
                }
        ...

    At this point holdCount reverts back to 0
```

**Critical Section**

- **unlocked** (holdCount = 0)
- **locked** (holdCount = 1)

T₁

T₂
Using ReentrantLock in Java

- ArrayBlockingQueue is a blocking bounded FIFO queue

```
public class ArrayBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>,
    java.io.Serializable {

    ... public E take() ... {
        final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
        try { ...
        } finally {
            lock.unlock();
        }
    }

    ... }

Ensure lock is always released when T₁ exits the critical section
```

See tutorials.jenkov.com/java-concurrency/locks.html#finally
Using ReentrantLock in Java

- ArrayBlockingQueue is a blocking bounded FIFO queue

```
public class ArrayBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, java.io.Serializable {
    ...
    public E take() ... {
        final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
    ...

    Thread T2 can now enter the critical section of take() & start running
```
Using ReentrantLock in Java

- ArrayBlockingQueue needs to use more than ReentrantLock to implement its semantics

```java
public class ArrayBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>,
    java.io.Serializable {

    ...

    public E take() ... {
        final ReentrantLock lock = this.lock;
        lock.lockInterruptibly();
        try {
            while (count == 0)
                notEmpty.await();
            return extract();
        } finally {
            lock.unlock();
        }
    }

    A Java ConditionObject is used to coordinate multiple threads
```

Upcoming lesson on “Java ConditionObject” shows more on ArrayBlockingQueue
• ArrayBlockingQueue needs to use more than ReentrantLock to implement its semantics

```
public class ArrayBlockingQueue<
    extends AbstractQueue<
    implements BlockingQueue<
    java.io.Serializable {

    ...

    public E take() ... {
        final ReentrantLock lock
            = this.lock;
        lock.lockInterruptibly();
        try {
            while (count == 0)
                notEmpty.await();
        return extract();
    } finally {
        lock.unlock();
    }

    These mechanisms implement Guarded Suspension & Monitor Object patterns
```

End of Java ReentrantLock (Part 4)