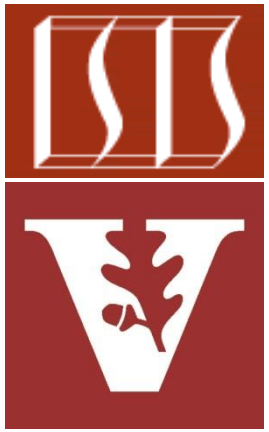
Java Concurrent Collections: Introduction

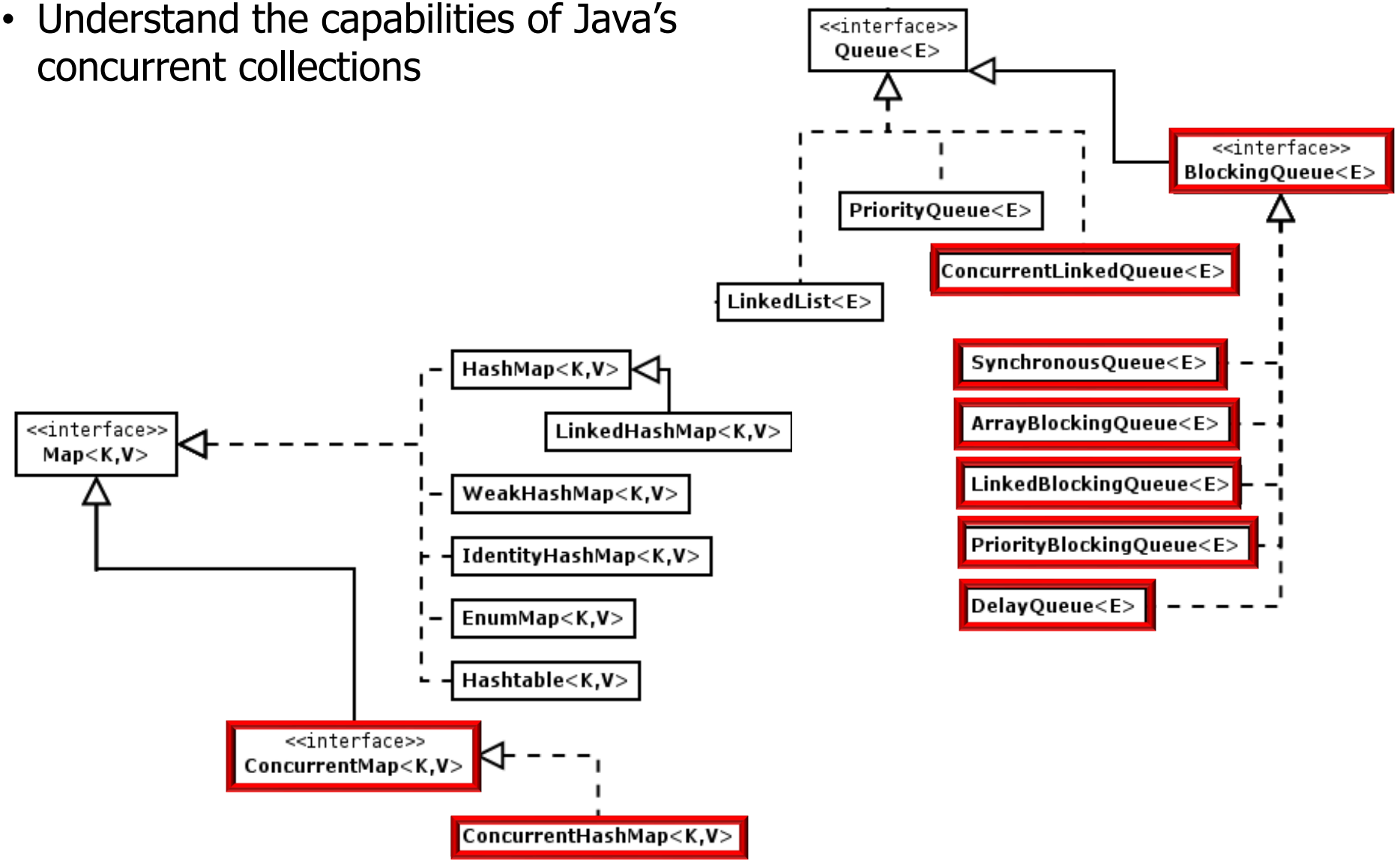


Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Lesson

- Understand the capabilities of Java's concurrent collections



Learning Objectives in this Lesson

- Understand the capabilities of Java's concurrent collections
  - As well as how Java's concurrent collections overcome limitations with Java's synchronized collections

Overview of Java Concurrent Collections

Overview of Java Concurrent Collections

- Java concurrent collections provide features that are optimized for the needs of concurrent programs

These are the concurrent-aware interfaces:

```
BlockingQueue
TransferQueue
BlockingDeque
ConcurrentMap
ConcurrentNavigableMap
```

Concurrent-aware classes include

```
LinkedBlockingQueue
ArrayBlockingQueue
PriorityBlockingQueue
DelayQueue
SynchronousQueue
LinkedBlockingDeque
LinkedTransferQueue
CopyOnWriteArrayList
CopyOnWriteArraySet
ConcurrentHashMap
```

See docs.oracle.com/javase/tutorial/essential/concurrency/collections.html

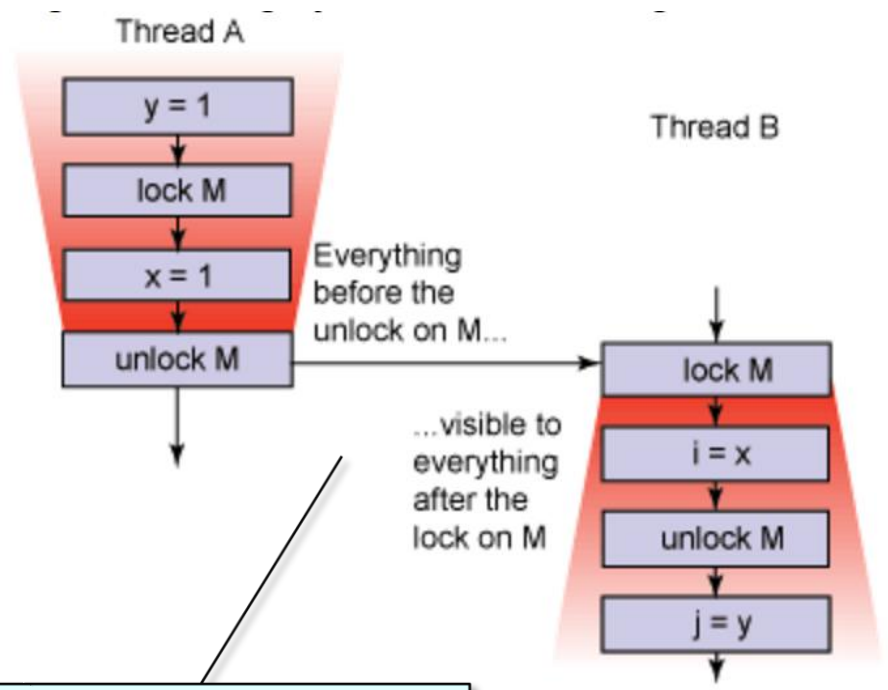
Overview of Java Concurrent Collections

- Java concurrent collections provide features that are optimized for the needs of concurrent programs
  - A concurrent collection is thread-safe, but is not governed by just a single exclusion lock

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html

Overview of Java Concurrent Collections

- Java concurrent collections provide features that are optimized for the needs of concurrent programs
  - A concurrent collection is thread-safe, but is not governed by just a single exclusion lock
  - They avoid *memory consistency errors* by defining a "happens-before" relationship

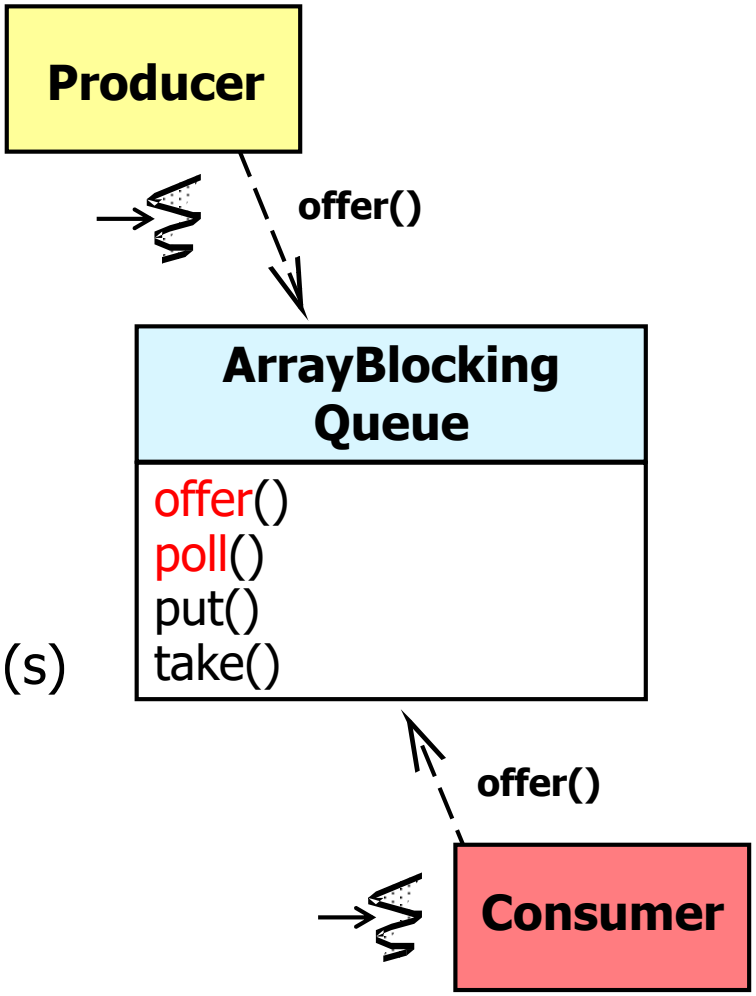


This relationship is a guarantee that memory writes by one specific statement are visible to another specific statement

See en.wikipedia.org/wiki/Happened-before

Overview of Java Concurrent Collections

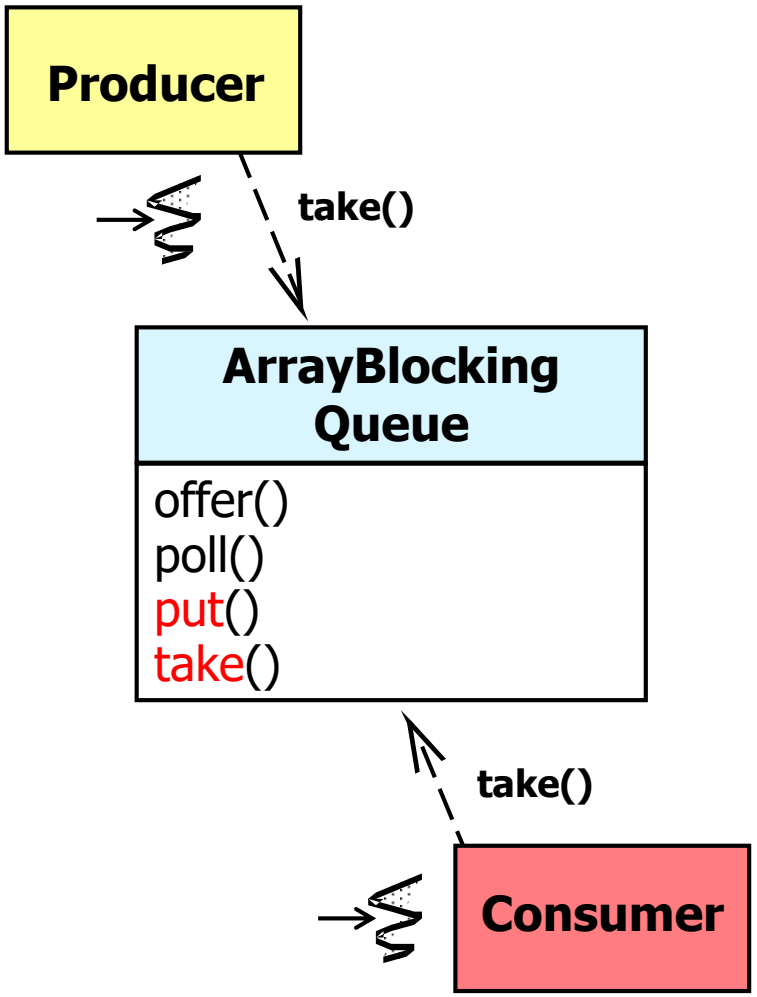
- Java concurrent collections provide features that are optimized for the needs of concurrent programs
  - A concurrent collection is thread-safe, but is not governed by just a single exclusion lock
  - They avoid *memory consistency errors* by defining a "happens-before" relationship
    - e.g., between a thread that adds an object to a collection with later thread(s) that access or remove that object



See docs.oracle.com/javase/tutorial/essential/concurrency/memconsist.html

Overview of Java Concurrent Collections

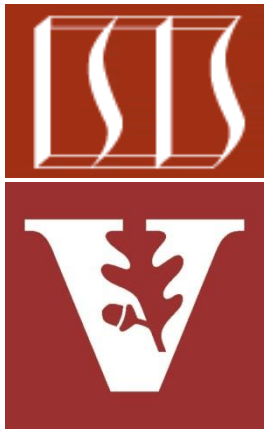
- Java concurrent collections provide features that are optimized for the needs of concurrent programs
  - A concurrent collection is thread-safe, but is not governed by just a single exclusion lock
  - They avoid *memory consistency errors* by defining a "happens-before" relationship
  - They enable needed blocking behavior on queues that are empty or full



See tutorials.jenkov.com/java-util-concurrent/blockingqueue.html

End of Java Concurrent Collections: Introduction

Java Concurrent Collections: ConcurrentHashMap & BlockingQueue



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 Lesson

- Understand the capabilities of Java's concurrent collections
- Recognize the capabilities of Java's ConcurrentHashMap & BlockingQueue

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.

Class ConcurrentHashMap<K,V>

java.lang.Object
  java.util.AbstractMap<K,V>
    java.util.concurrent.ConcurrentHashMap<K,V>

Type Parameters:

`K - the type of keys maintained by this map`

`V - the type of mapped values`

All Implemented Interfaces:

`Serializable, ConcurrentMap<K,V>, Map<K,V>`

```
public class ConcurrentHashMap<K,V>
extends AbstractMap<K,V>
implements ConcurrentMap<K,V>, Serializable
```

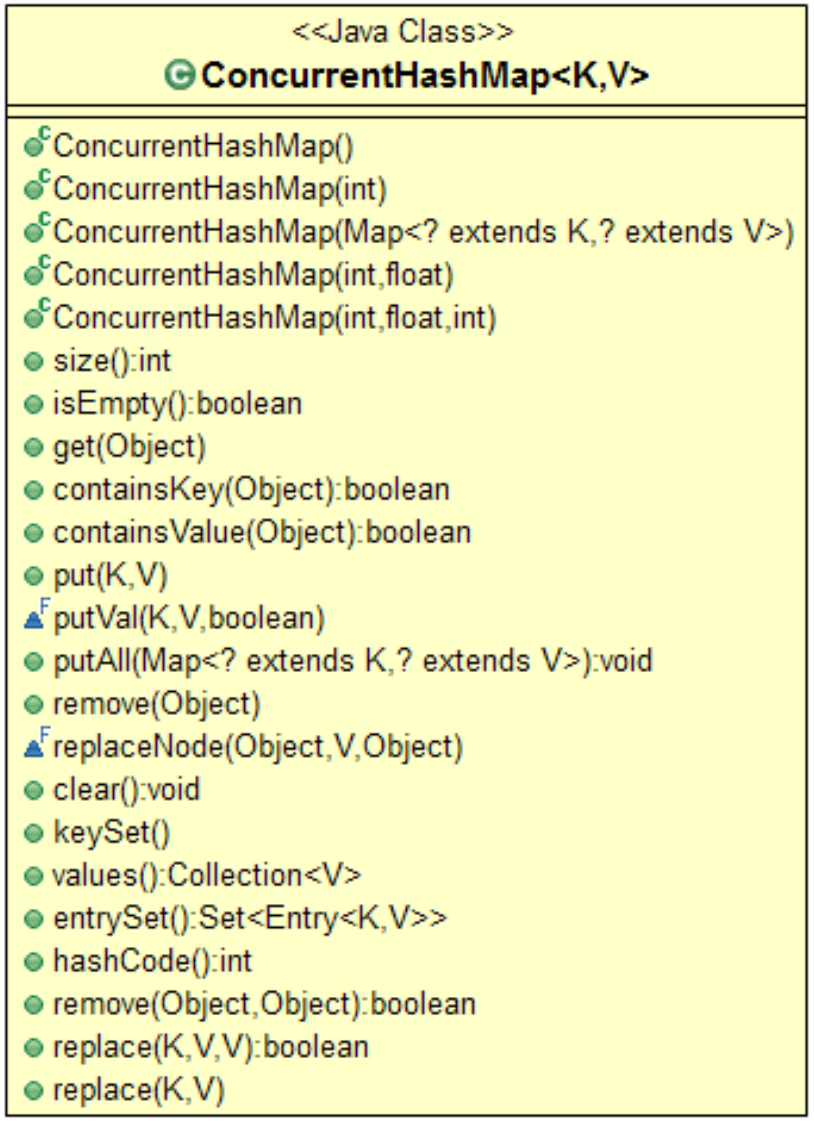
A hash table supporting full concurrency of retrievals and high expected concurrency for updates. This class obeys the same functional specification as Hashtable, and includes versions of methods corresponding to each method of Hashtable. However, even though all operations are thread-safe, retrieval operations do *not* entail locking, and there is *not* any support for locking the entire table in a way that prevents all access. This class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details.

Retrieval operations (including get) generally do not block, so may overlap with update operations (including put and remove). Retrievals reflect the results of the most recently *completed* update operations holding upon their onset. (More formally, an update operation for a given key bears a *happens-before* relation with any (non-null) retrieval for that key reporting the

Overview of Java ConcurrentHashMap

Overview of Java ConcurrentHashMap

- Enables concurrent retrievals & adjustable expected concurrent updates via OO & functional programming APIs



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html

Overview of Java ConcurrentHashMap

- Optimized for multi-core CPUs

Building a better HashMap

How ConcurrentHashMap offers higher concurrency without compromising thread safety

Brian Goetz
Published on August 21, 2003

4

Content series:

+ This content is part of the series: **Java theory and practice**

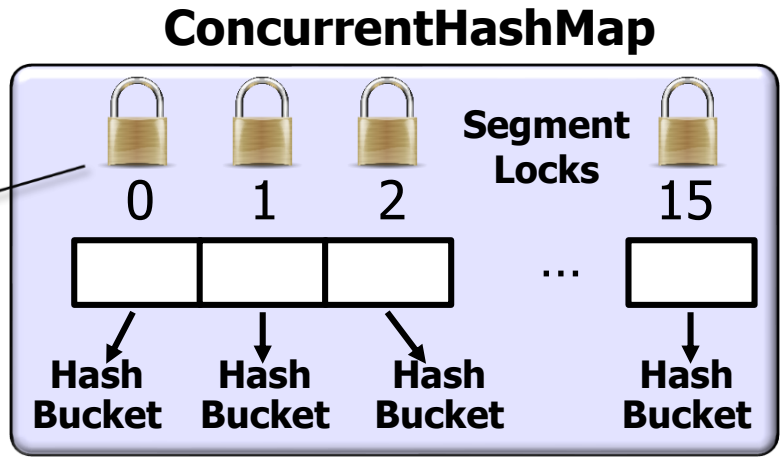
In July's installment of *Java theory and practice* ("Concurrent collections classes"), we reviewed scalability bottlenecks and discussed how to achieve higher concurrency and throughput in shared data structures. Sometimes, the best way to learn is to examine the work of the experts, so this month we're going to look at the implementation of `ConcurrentHashMap` from Doug Lea's `util.concurrent` package. A version of `ConcurrentHashMap` optimized for the new Java Memory Model (JMM), which is being specified by JSR 133, will be included in the `java.util.concurrent` package in JDK 1.5; the version in `util.concurrent` has been audited for thread-safety under both the old and new memory models.

See www.ibm.com/developerworks/library/j-jtp08223

Overview of Java ConcurrentHashMap

- Optimized for multi-core CPUs
  - It uses a group of locks, each guarding a subset of hash buckets

These segment locks minimize contention

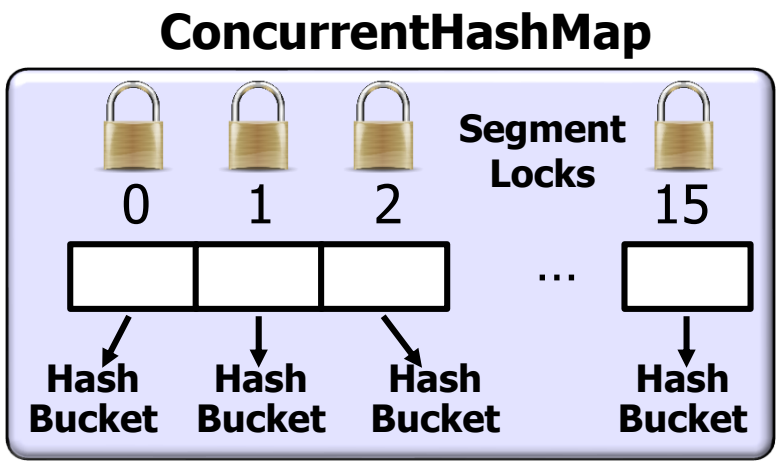


See codepumpkin.com/hashtable-vs-synchronizedmap-vs-concurrenthashmap

Overview of Java ConcurrentHashMap

- Optimized for multi-core CPUs
  - It uses a group of locks, each guarding a subset of hash buckets

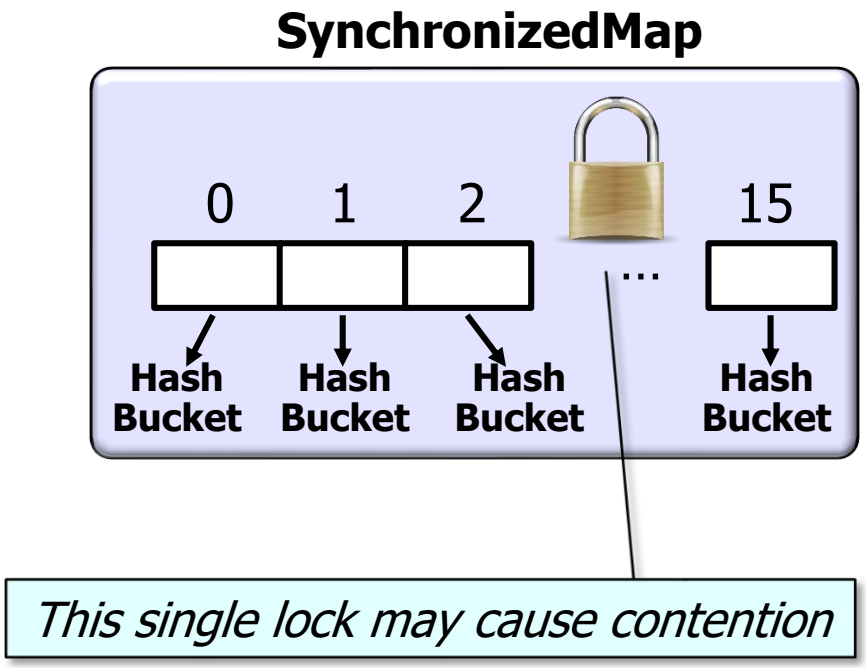
There are common human known uses!



See codepumpkin.com/hashtable-vs-synchronizedmap-vs-concurrenthashmap

Overview of Java ConcurrentHashMap

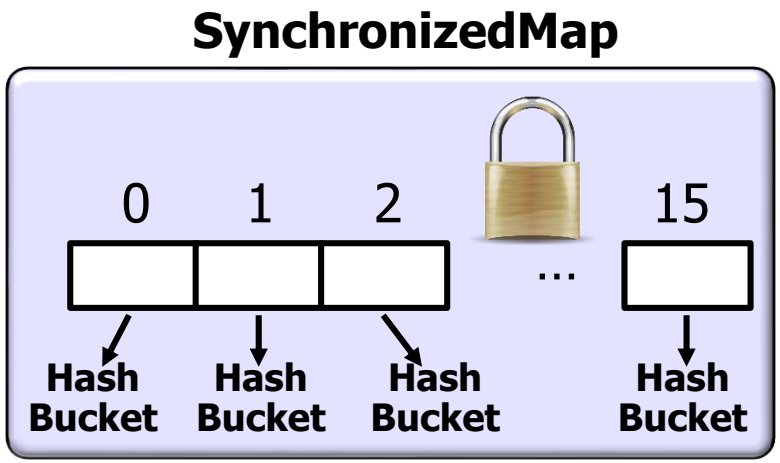
- Optimized for multi-core CPUs
  - It uses a group of locks, each guarding a subset of hash buckets
  - Conversely, a SynchronizedMap only uses a single lock



See codepumpkin.com/hashtable-vs-synchronizedmap-vs-concurrenthashmap

Overview of Java ConcurrentHashMap

- Optimized for multi-core CPUs
  - It uses a group of locks, each guarding a subset of hash buckets
  - Conversely, a SynchronizedMap only uses a single lock

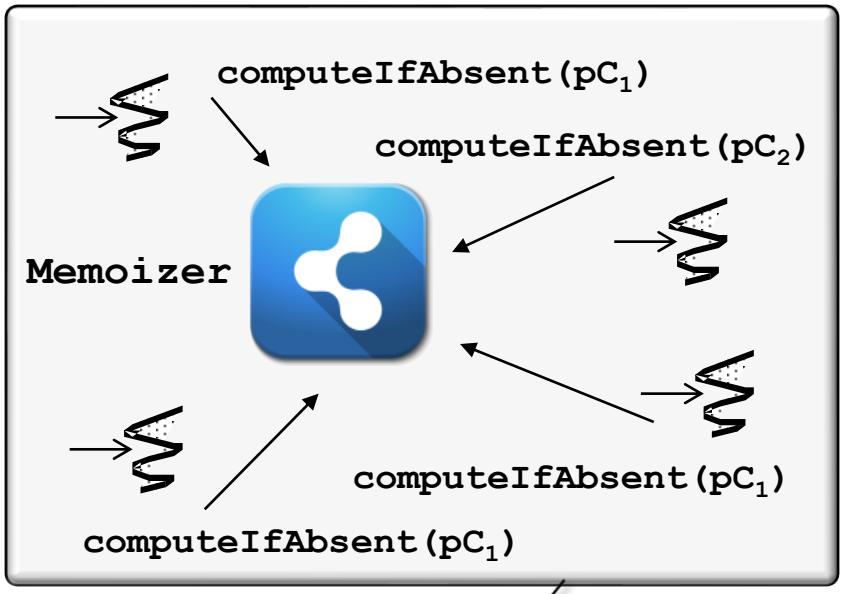


There are also common human known uses of this approach!

See codepumpkin.com/hashtable-vs-synchronizedmap-vs-concurrenthashmap

Overview of Java ConcurrentHashMap

- Provides "atomic check-then-act" methods



Only one computation per key is performed even if multiple threads call computeIfAbsent() using the same key

See dig.cs.illinois.edu/papers/checkThenAct.pdf

Overview of Java ConcurrentHashMap

- Provides "atomic check-then-act" methods, e.g.
  - If key isn't already associated w/a value, compute its value using the given function & enter it into map

Instead of

```
V value = map.get(key);
if (value == null) {
  value =
    mappingFunc.apply(key);
  if (value != null)
    map.put(key, value);
}
return value;
```

use

```
return map.computeIfAbsent
  (key, k -> new Value(f(k)));
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#computeIfAbsent

Overview of Java ConcurrentHashMap

- Provides "atomic check-then-act" methods, e.g.
  - If key isn't already associated w/a value, compute its value using the given function & enter it into map
  - If a key isn't already associated w/a value, associate it with the value

Instead of

```
V value = map.get(key);
if (value == null)
  return map.put(key, value);
else
  return value;
```

use

```
return map.putIfAbsent
                (key, value);
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#putIfAbsent

Overview of Java ConcurrentHashMap

- Provides "atomic check-then-act" methods, e.g.
  - If key isn't already associated w/a value, compute its value using the given function & enter it into map
  - If a key isn't already associated w/a value, associate it with the value
  - Replaces entry for a key only if currently mapped to some value

Instead of

```
if (map.containsKey(key))
  return map.put(key, value);
else
  return null;
```

use

```
return map.replace(key, value);
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#replace

Overview of Java ConcurrentHashMap

- Provides "atomic check-then-act" methods, e.g.
  - If key isn't already associated w/a value, compute its value using the given function & enter it into map
  - If a key isn't already associated w/a value, associate it with the value
  - Replaces entry for a key only if currently mapped to some value
  - Replaces entry for a key only if currently mapped to given value

Instead of

```
if (map.containsKey(key) &&
    Objects.equals(map.get(key),
                   oldValue)) {
   map.put(key, newValue);
   return true;
} else
  return false;
```

use

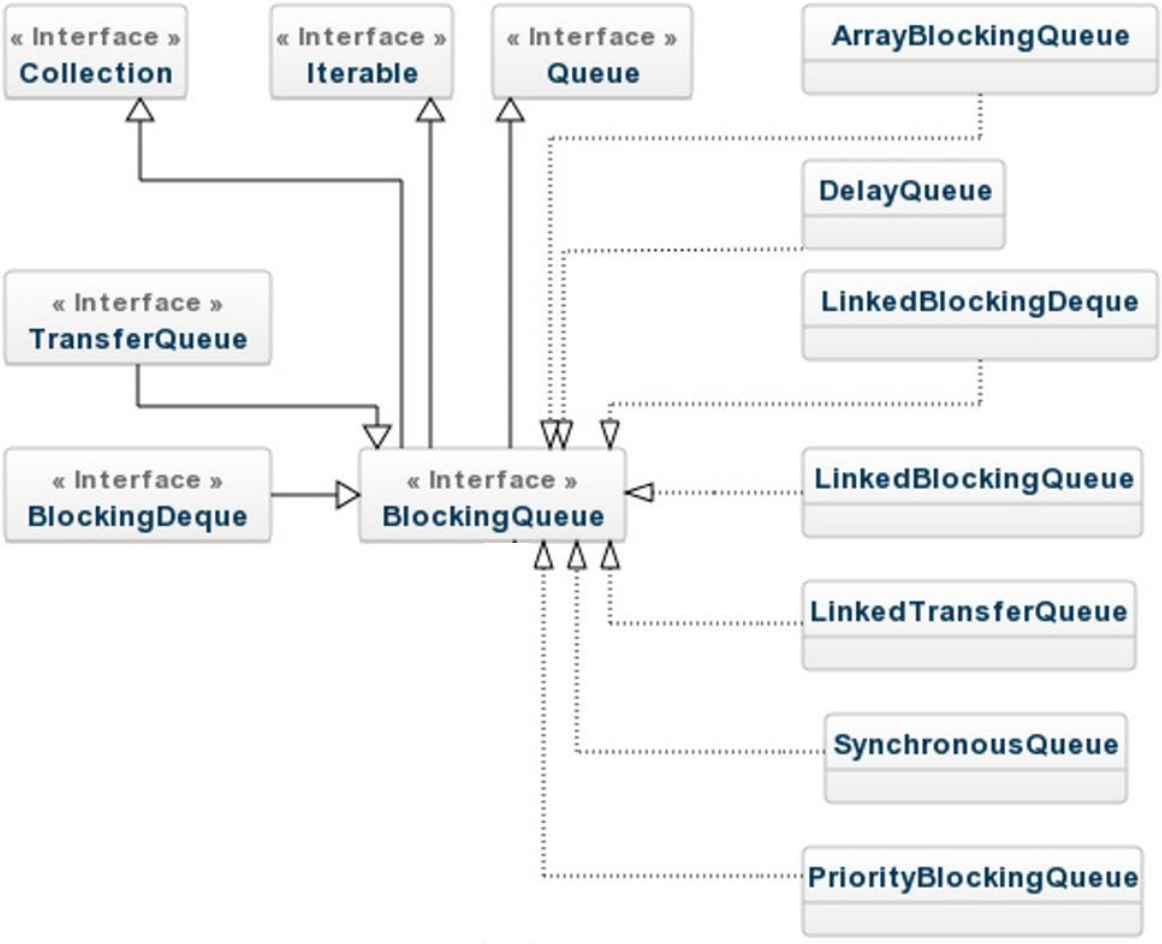
```
return map.replace(key,
                   oldValue,
                   newValue);
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#replace

Overview of Java BlockingQueue

Overview of Java BlockingQueue

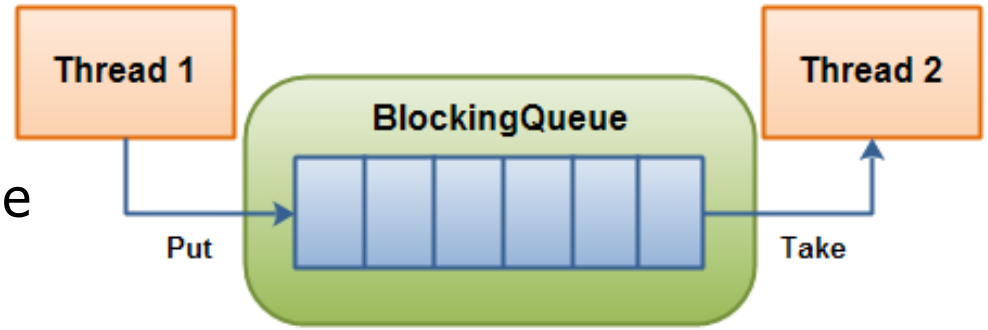
- A Queue supporting operations can wait for the queue to become non-empty when retrieving an element & wait for space to become available in queue when storing an element



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html

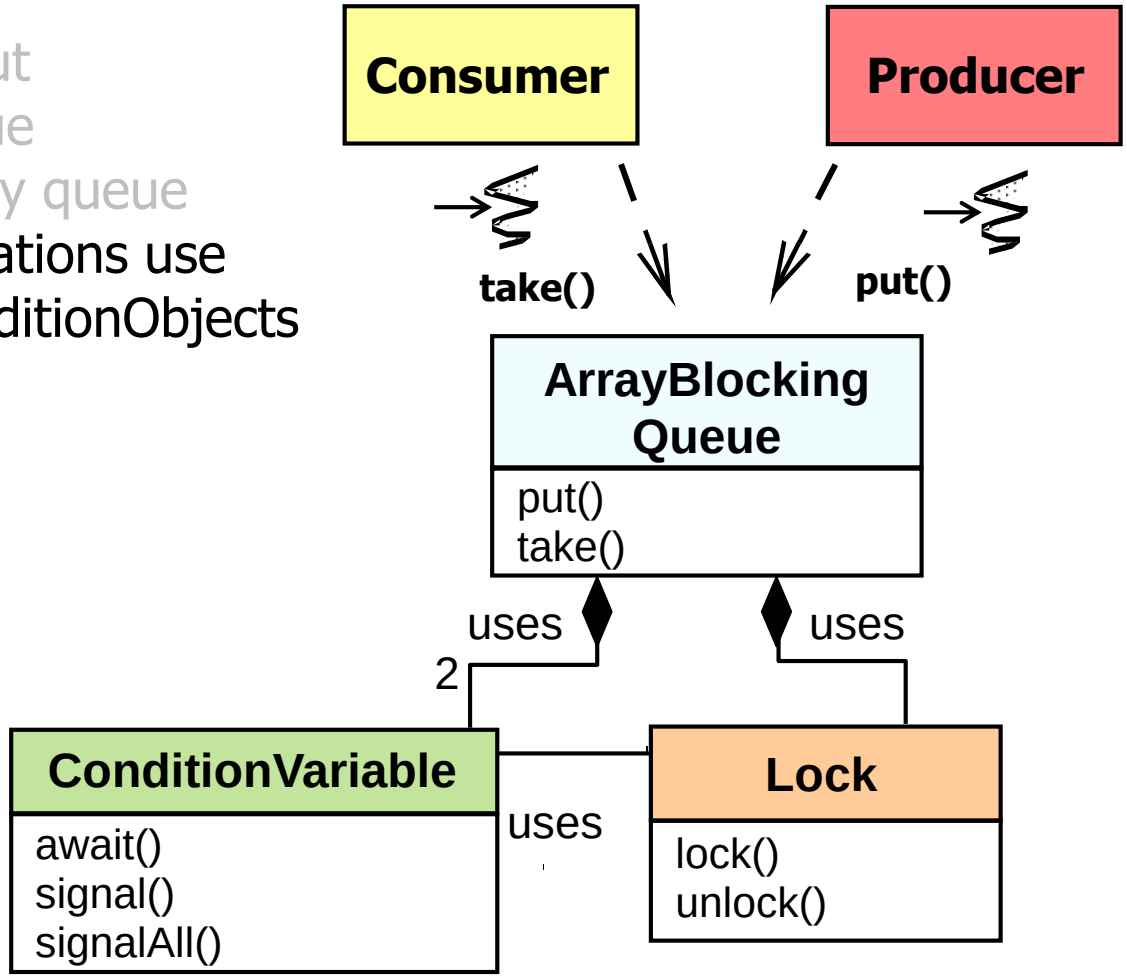
Overview of Java BlockingQueue

- A Queue supporting operations can wait for the queue to become non-empty when retrieving an element & wait for space to become available in queue when storing an element
  - Clients can block or timeout when adding to a full queue or retrieving from an empty queue



Overview of Java BlockingQueue

- A Queue supporting operations can wait for the queue to become non-empty when retrieving an element & wait for space to become available in queue when storing an element
  - Clients can block or timeout when adding to a full queue or retrieving from an empty queue
  - BlockingQueue implementations use Java ReentrantLock & ConditionObjects



See earlier lessons on "*Java ReentrantLock*" & "*Java ConditionObject*"

End of Java Concurrent Collections: ConcurrentHash Map & BlockingQueue