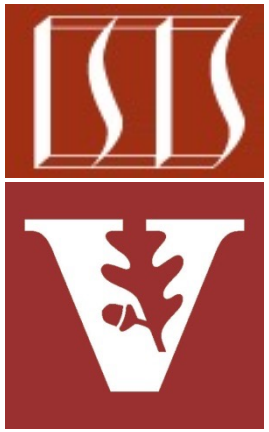
Types of Java Synchronizer Capabilities (Part 1)



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

- Be aware of the Java memory model
- Understand the purpose of Java synchronizers
- Recognize the pervasiveness of Java synchronizers
- Know the types of capabilities provided by Java synchronizers

Category	Definition
Atomic operations	An action that effectively happens all at once or not at all
Mutual exclusion	Allows concurrent access & updates to shared mutable data without race conditions
Coordination	Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.
Barrier synchronization	Ensures that any thread(s) must stop at a certain point & cannot proceed until all other thread(s) reach this barrier

Learning Objectives in this Part of the Lesson

- Be aware of the Java memory model
- Understand the purpose of Java synchronizers
- Recognize the pervasiveness of Java synchronizers
- Know the types of capabilities provided by Java synchronizers

Category	Definition
Atomic operations	An action that effectively happens all at once or not at all
Mutual exclusion	Allows concurrent access & updates to shared mutable data without race conditions
Coordination	Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.
Barrier synchronization	Ensures that any thread(s) must stop at a certain point & cannot proceed until all other thread(s) reach this barrier

Learning Objectives in this Part of the Lesson

- Be aware of the Java memory model
- Understand the purpose of Java synchronizers
- Recognize the pervasiveness of Java synchronizers
- Know the types of capabilities provided by Java synchronizers

Category	Definition
Atomic operations	An action that effectively happens all at once or not at all
Mutual exclusion	Allows concurrent access & updates to shared mutable data without race conditions
Coordination	Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.
Barrier synchronization	Ensures that any thread(s) must stop at a certain point & cannot proceed until all other thread(s) reach this barrier

Types of Java Synchronizer Capabilities

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities

Category	Definition
Atomic operations	An action that effectively happens all at once or not at all
Mutual exclusion	Allows concurrent access & updates to shared mutable data without race conditions
Coordination	Ensures computations run properly, e.g., in the right order, at the right time, under the right conditions, etc.
Barrier synchronization	Ensures that any thread(s) must stop at a certain point & cannot proceed until all other thread(s) reach this barrier

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - **Atomic ordering**
    - Ensures an action happens all at once or not at all

See en.wikipedia.org/wiki/Linearizability

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - **Atomic ordering**
    - Ensures an action happens all at once or not at all
    - Operations on a field in thread$_1$ occur all at once wrt operations on the field in thread$_{2..n}$

time ↓

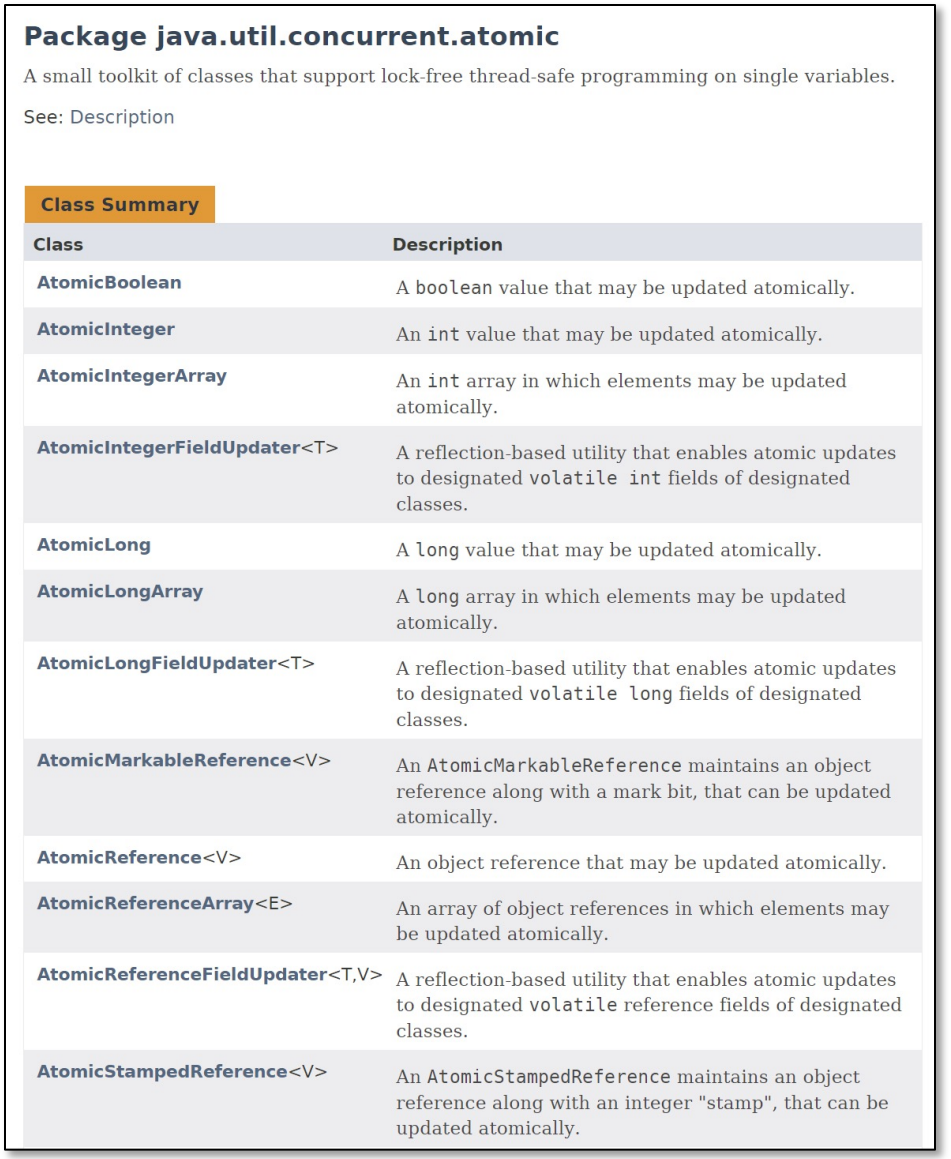
Thread$_1$	Thread$_2$		Long field
initialized			0
read field		←	0
increase field by 1			0
write back		→	1
	read field	←	1
	increase field by 1		1
	write back	→	2

Atomicity does not occur on primitive Java data types without using synchronizers

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

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - **Atomic ordering**
    - Ensures an action happens all at once or not at all
    - Operations on a field in thread$_1$ occur all at once wrt operations on the field in thread$_{2..n}$
    - Atomic ordering is supported by the Java atomic package



Package java.util.concurrent.atomic

A small toolkit of classes that support lock-free thread-safe programming on single variables.

See: Description

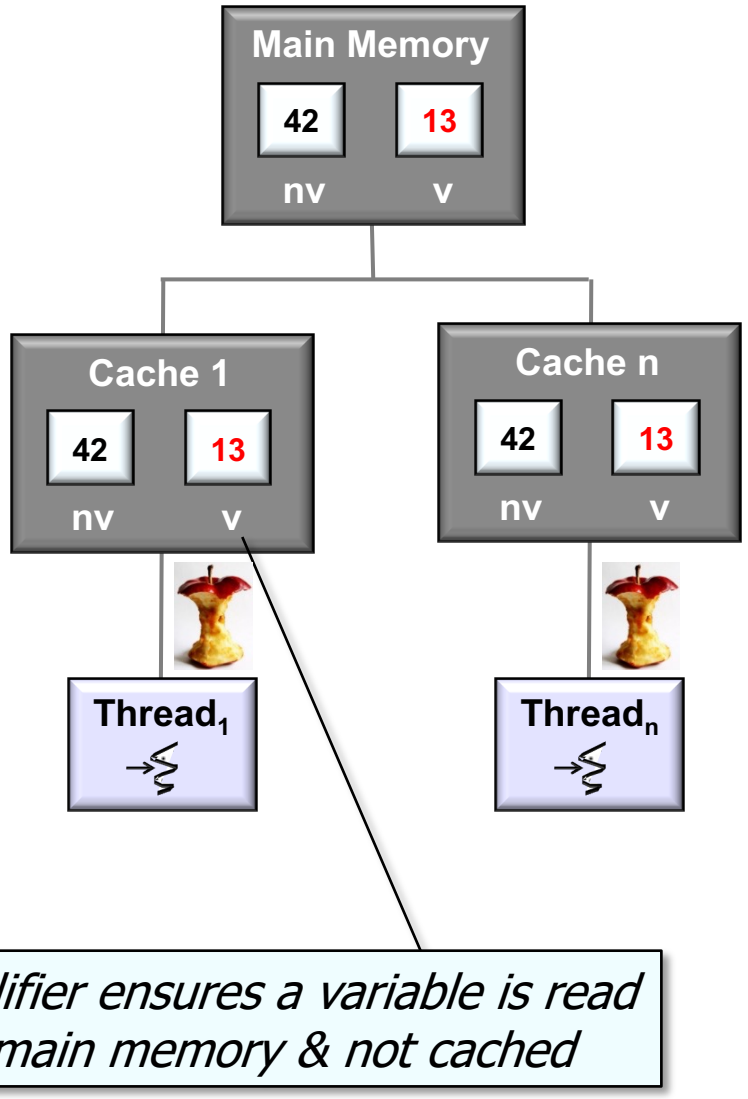
Class Summary

Class	Description
AtomicBoolean	A `boolean` value that may be updated atomically.
AtomicInteger	An `int` value that may be updated atomically.
AtomicIntegerArray	An `int` array in which elements may be updated atomically.
AtomicIntegerFieldUpdater<T>	A reflection-based utility that enables atomic updates to designated `volatile int` fields of designated classes.
AtomicLong	A `long` value that may be updated atomically.
AtomicLongArray	A `long` array in which elements may be updated atomically.
AtomicLongFieldUpdater<T>	A reflection-based utility that enables atomic updates to designated `volatile long` fields of designated classes.
AtomicMarkableReference<V>	An `AtomicMarkableReference` maintains an object reference along with a mark bit, that can be updated atomically.
AtomicReference<V>	An object reference that may be updated atomically.
AtomicReferenceArray<E>	An array of object references in which elements may be updated atomically.
AtomicReferenceFieldUpdater<T,V>	A reflection-based utility that enables atomic updates to designated `volatile` reference fields of designated classes.
AtomicStampedReference<V>	An `AtomicStampedReference` maintains an object reference along with an integer "stamp", that can be updated atomically.

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

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - **Atomic ordering**
    - Ensures an action happens all at once or not at all
    - Operations on a field in $thread_1$ occur all at once wrt operations on the field in $thread_{2..n}$
    - Atomic ordering is supported by the Java atomic package
  - Atomic ordering is also supported by the Java volatile type qualifier



See en.wikipedia.org/wiki/Volatile_variable#In_Java

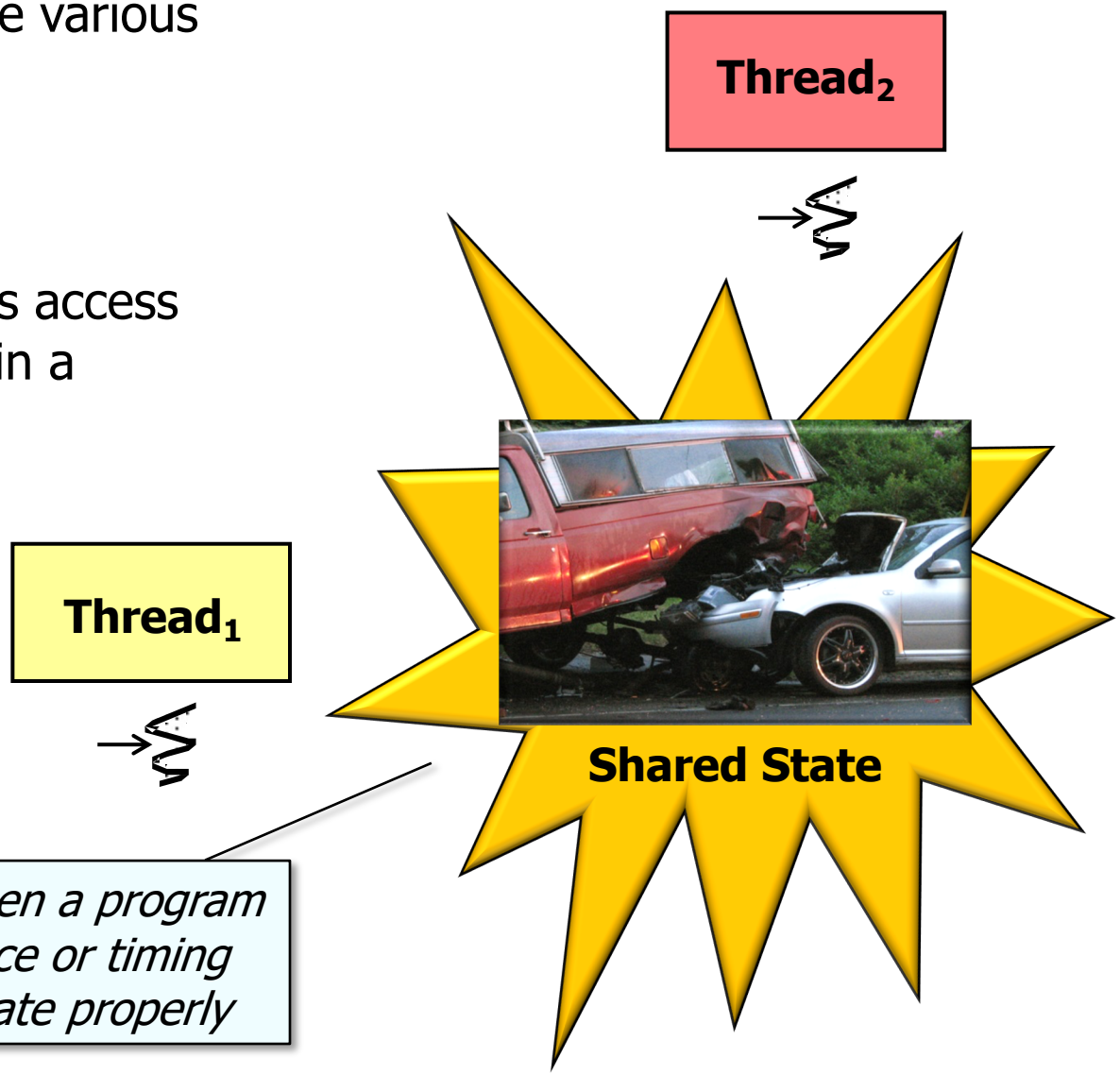
Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - **Mutual exclusion**
    - Prevents simultaneous access to a shared resource in a critical section

See en.wikipedia.org/wiki/Mutual_exclusion

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - **Mutual exclusion**
    - Prevents simultaneous access to a shared resource in a critical section



Race conditions occur when a program depends on the sequence or timing of threads for it to operate properly

See en.wikipedia.org/wiki/Race_condition#Software

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - **Mutual exclusion**
    - Prevents simultaneous access to a shared resource in a critical section
    - Read/write conflicts
      - If one thread reads while another thread writes concurrently, the field that's read may be inconsistent

time ↓

$Thread_1$	$Thread_2$		Long field
initialized			0
read field		←	0
increase field by 1			0
write back	read field	← →	0 or 1?

Two operations conflict if at least one is a write

See en.wikipedia.org/wiki/Read-write_conflict

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - **Atomic ordering**
  - **Mutual exclusion**
    - Prevents simultaneous access to a shared resource in a critical section
    - Read/write conflicts
    - Write/write conflicts
      - If two threads try to write to same field concurrently, the result may be inconsistent

time ↓

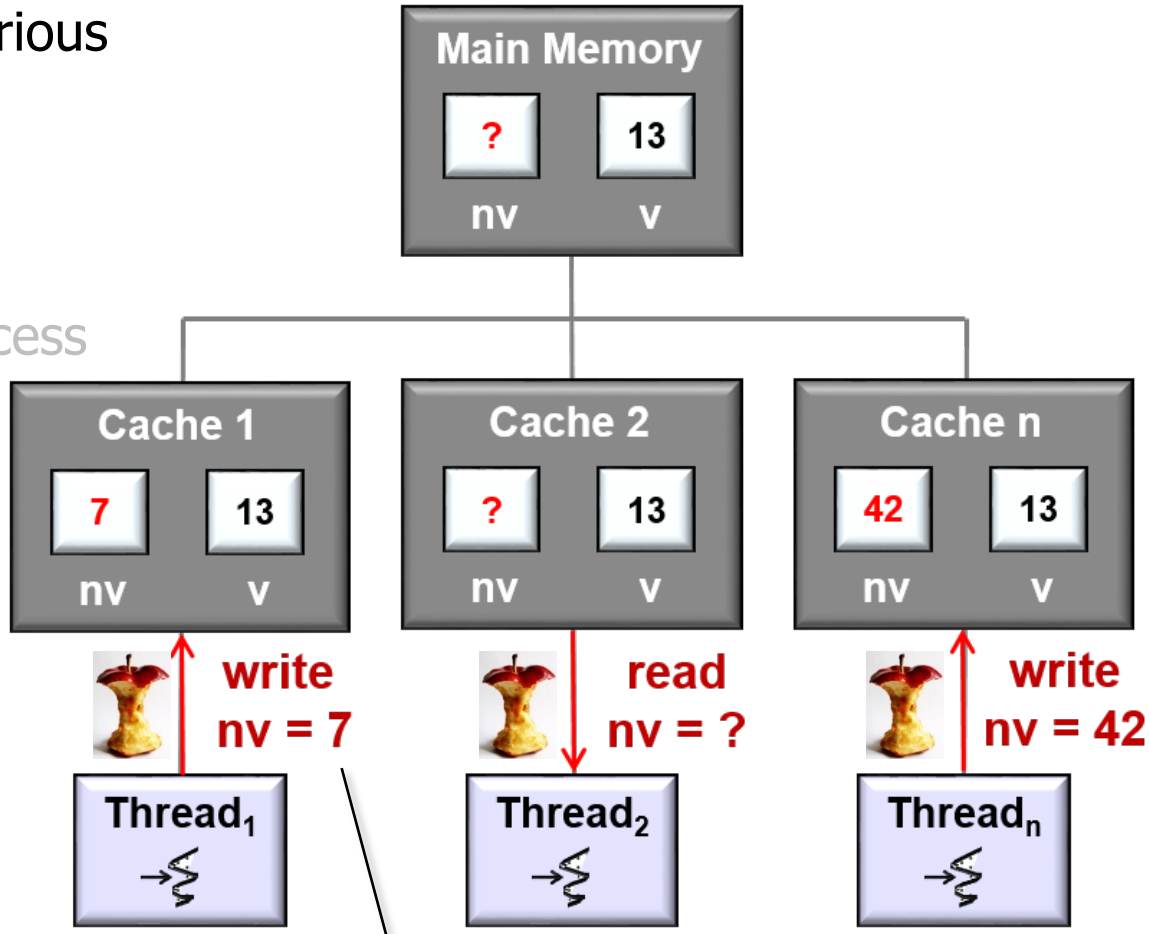
$Thread_1$	$Thread_2$		Long field
initialized			0
read field		←	0
	read field	←	0
increase field by 2			0
	increase field by 1		0
write back	write back	→	1 or 2?

This can yield a "lost update"

See en.wikipedia.org/wiki/Write-write_conflict

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - **Mutual exclusion**
    - Prevents simultaneous access to a shared resource in a critical section
    - Read/write conflicts
    - Write/write conflicts



These problems often occur in multi-core processors with "weak" memory ordering due to core caches that allow "out-of-order" load & store operations

See en.wikipedia.org/wiki/Memory_ordering

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - **Mutual exclusion**
    - Prevents simultaneous access to a shared resource in a critical section
    - Read/write conflicts
    - Write/write conflicts
    - Mutual exclusion is supported by the Java locks package
      - e.g., ReentrantLock, ReentrantReadWriteLock, StampedLock, etc.

Package java.util.concurrent.locks

Interfaces and classes providing a framework for locking and waiting for conditions that is distinct from built-in synchronization and monitors.

See: Description

Interface Summary

Interface	Description
Condition	`Condition` factors out the `Object` monitor methods (`wait`, `notify` and `notifyAll`) into distinct objects to give the effect of having multiple wait-sets per object, by combining them with the use of arbitrary `Lock` implementations.
Lock	`Lock` implementations provide more extensive locking operations than can be obtained using `synchronized` methods and statements.
ReadWriteLock	A `ReadWriteLock` maintains a pair of associated `locks`, one for read-only operations and one for writing.

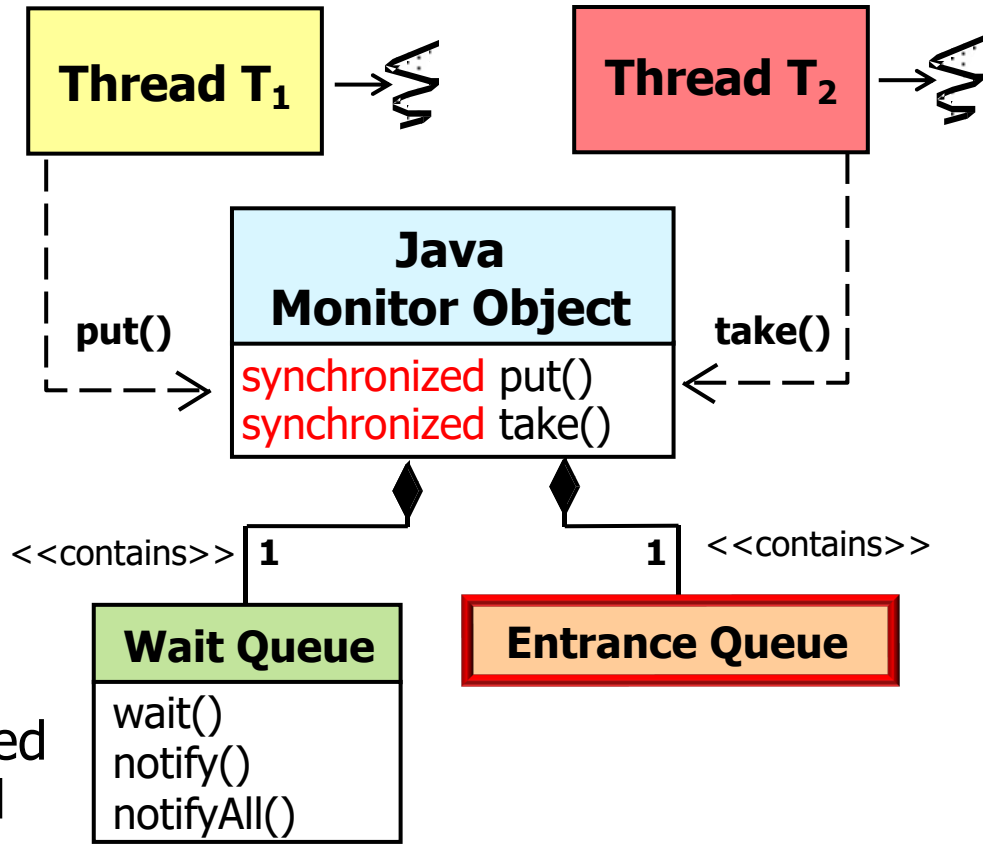
Class Summary

Class	Description
AbstractOwnableSynchronizer	A synchronizer that may be exclusively owned by a thread.
AbstractQueuedLongSynchronizer	A version of `AbstractQueuedSynchronizer` in which synchronization state is maintained as a `long`.
AbstractQueuedSynchronizer	Provides a framework for implementing blocking locks and related synchronizers (semaphores, events, etc) that rely on first-in-first-out (FIFO) wait queues.
LockSupport	Basic thread blocking primitives for creating locks and other synchronization classes.
ReentrantLock	A reentrant mutual exclusion `Lock` with the same basic behavior and semantics as the implicit monitor lock accessed using `synchronized` methods and statements, but with extended capabilities.
ReentrantReadWriteLock	An implementation of `ReadWriteLock` supporting similar semantics to `ReentrantLock`.

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

Types of Java Synchronizer Capabilities

- Java synchronizers provide various types of capabilities, e.g.
  - Atomic ordering
  - **Mutual exclusion**
    - Prevents simultaneous access to a shared resource in a critical section
    - Read/write conflicts
    - Write/write conflicts
    - Mutual exclusion is supported by the Java locks package
  - Mutual exclusion is also supported by the **`synchronized`** keyword in Java built-in monitor objects



See www.artima.com/insidejvm/ed2/threadsynch.html

End of Types of Java Synchronizer Capabilities (Part 1)