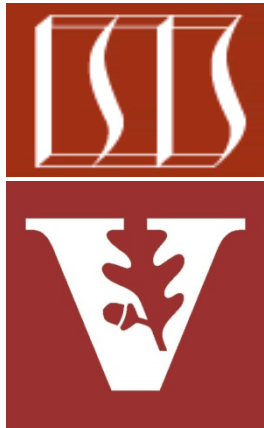


Synopsis of Java Synchronizer Classes



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
- Learn the key synchronizers defined in the Java class library

Java Class	Purpose
ReentrantLock	A reentrant mutual exclusion lock that extends the built-in monitor lock capabilities
ReentrantReadWriteLock	Improves performance when resources are read much more often than written
StampedLock	A readers-writer lock that's more efficient than ReentrantReadWriteLock
Semaphore	Maintains permits that controls thread access to limited # of shared resources
ConditionObject	Allows Thread to block until a condition becomes true
CountDownLatch	Allows one or more threads to wait until a set of operations being performed in other threads complete
CyclicBarrier	Allows a set of threads to all wait for each other to reach a common barrier point
Phaser	A more flexible reusable synchronization barrier

Overview of Java Synchronizer Classes

Overview of Java Synchronizer Classes

- The Java class library defines *many* synchronizers
 - e.g., `java.util.concurrent` & `java.util.concurrent.locks` packages

package

Added in API level 1

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. The framework permits much greater flexibility in the use of locks and conditions, at the expense of more awkward syntax.

The `Lock` interface supports locking disciplines that differ in semantics (reentrant, fair, etc), and that can be used in non-block-structured contexts including hand-over-hand and lock reordering algorithms. The main implementation is `ReentrantLock`.

package

Added in API level 1

java.util.concurrent

Utility classes commonly useful in concurrent programming. This package includes a few small standardized extensible frameworks, as well as some classes that provide useful functionality and are otherwise tedious or difficult to implement. Here are brief descriptions of the main components. See also the `java.util.concurrent.locks` and `java.util.concurrent.atomic` packages.

See developer.android.com/reference/java/util/concurrent/package-summary.html

Overview of Java Synchronizer Classes

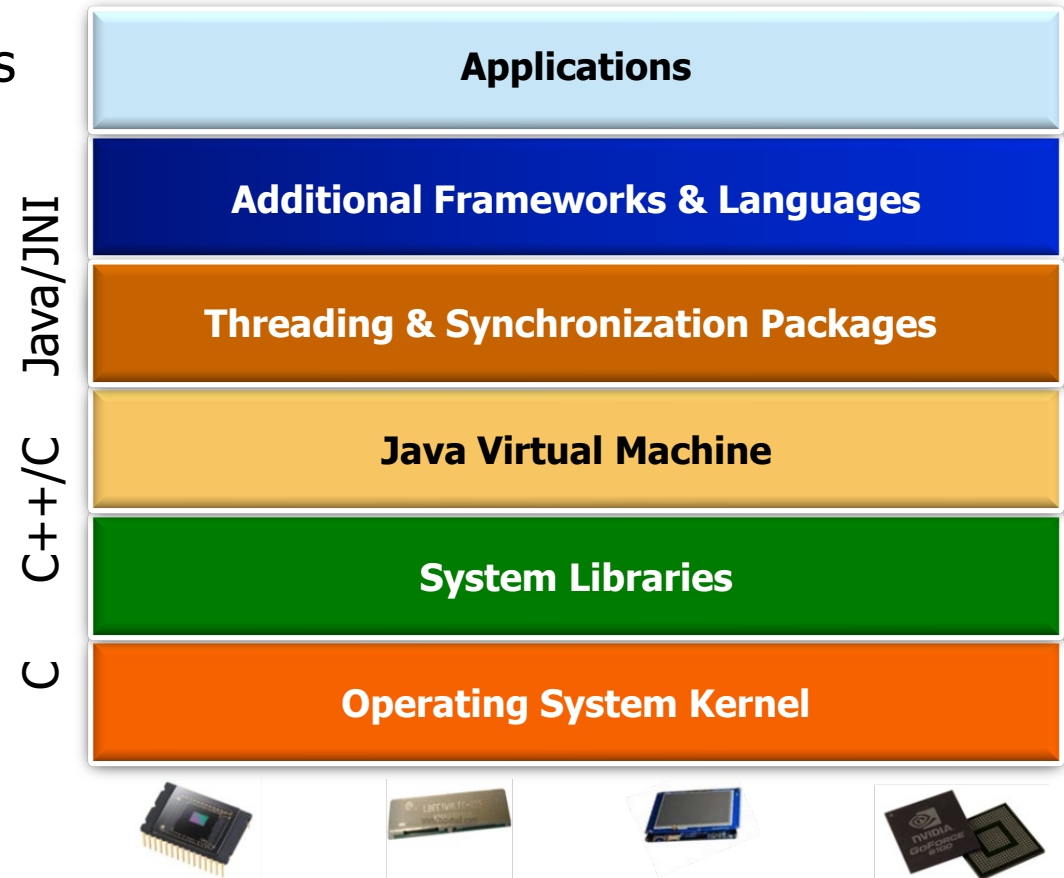
- We cover Java language features & library classes for synchronization

Java Class	Purpose
ReentrantLock	A reentrant mutual exclusion lock that extends the built-in monitor lock capabilities
ReentrantReadWriteLock	Improves performance when resources are read much more often than written
StampedLock	A readers-writer lock that's more efficient than ReentrantReadWriteLock
Semaphore	Maintains permits that control thread access to limited # of shared resources
ConditionObject	Allows Thread to block until a condition becomes true
CountDownLatch	Allows one or more Threads to wait until a set of operations being performed in other Threads complete
CyclicBarrier	Allows a set of Threads to all wait for each other to reach a common barrier point
Phaser	A more flexible reusable synchronization barrier

We show how these features & classes are implemented & used in Java & in practice

Overview of Java Synchronizer Classes

- These synchronizers are used extensively in Java applications & class libraries



Overview of Java Synchronizer Classes























- **ReentrantLock**

- A mutual exclusion lock that extends built-in monitor lock capabilities



<<Java Class>>

ReentrantLock

-  ReentrantLock()
-  ReentrantLock(boolean)
-  lock():void
-  lockInterruptibly():void
-  tryLock():boolean
-  tryLock(long,TimeUnit):boolean
-  unlock():void
-  newCondition():Condition
-  getHoldCount():int
-  isHeldByCurrentThread():boolean
-  isLocked():boolean
-   isFair():boolean
-   hasQueuedThreads():boolean
-   hasQueuedThread(Thread):boolean
-   getQueueLength():int
-  hasWaiters(Condition):boolean
-  getWaitQueueLength(Condition):int
-  toString()

Overview of Java Synchronizer Classes

- **ReentrantLock**

- A mutual exclusion lock that extends built-in monitor lock capabilities
- “Reentrant” means that the thread holding the lock can reacquire it without deadlock



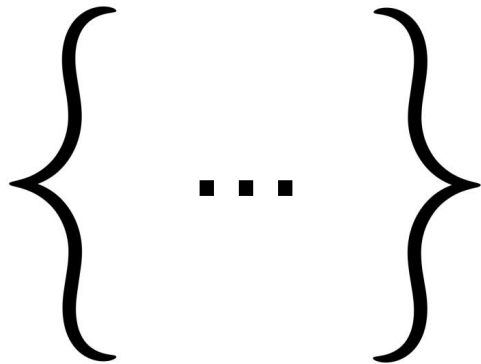
<<Java Class>>	
G ReentrantLock	
● ^F	ReentrantLock()
● ^F	ReentrantLock(boolean)
●	lock():void
●	lockInterruptibly():void
●	tryLock():boolean
●	tryLock(long,TimeUnit):boolean
●	unlock():void
●	newCondition():Condition
●	getHoldCount():int
●	isHeldByCurrentThread():boolean
●	isLocked():boolean
● ^F	isFair():boolean
● ^F	hasQueuedThreads():boolean
● ^F	hasQueuedThread(Thread):boolean
● ^F	getQueueLength():int
●	hasWaiters(Condition):boolean
●	getWaitQueueLength(Condition):int
●	toString()

See [en.wikipedia.org/wiki/Reentrancy_\(computing\)](https://en.wikipedia.org/wiki/Reentrancy_(computing))

Overview of Java Synchronizer Classes

• ReentrantLock

- A mutual exclusion lock that extends built-in monitor lock capabilities
- “Reentrant” means that the thread holding the lock can reacquire it without deadlock
- Must be “fully bracketed”
 - A thread that acquires a lock must be the one to release it



<<Java Class>>	
G ReentrantLock	
●	ReentrantLock()
●	ReentrantLock(boolean)
●	lock():void
●	lockInterruptibly():void
●	tryLock():boolean
●	tryLock(long,TimeUnit):boolean
●	unlock():void
●	newCondition():Condition
●	getHoldCount():int
●	isHeldByCurrentThread():boolean
●	isLocked():boolean
● ^F	isFair():boolean
● ^F	hasQueuedThreads():boolean
● ^F	hasQueuedThread(Thread):boolean
● ^F	getQueueLength():int
●	hasWaiters(Condition):boolean
●	getWaitQueueLength(Condition):int
●	toString()

Overview of Java Synchronizer Classes

- **ReentrantReadWriteLock**

- Improves performance when resources read more often than written

UNDERGRADUATE CATALOG
VANDERBILT UNIVERSITY

2014/2015

Containing general information and courses of study for the 2014/2015 session corrected to 18 June 2014

Download a pdf file of the [Undergraduate Catalog](#) (15.2 MB)

View specific sections of the catalog below

- [Contents](#)
- [Calendar](#)
- [The University](#)
- [Vanderbilt University Board of Trust](#)
- [Vanderbilt University Administration](#)
- [Special Programs for Undergraduates](#)
- [Life at Vanderbilt](#)
- [Admission](#)
- [Financial Information](#)
- [Scholarships and Need-Based Financial Aid](#)
- [College of Arts & Science](#)
- [Blair School of Music](#)
- [School of Engineering](#)
- [Peabody College](#)
- [Index](#)

This is the online version of the Undergraduate Catalog, a printed document of record issued in the fall of each year. The online version mirrors the actual printed book and is not updated until the new edition is printed for each year. For ongoing updates to departmental information, go to the website of the individual department.

ABOUT THE PDFs
Please note the catalogs listed below are in pdf format. You will need Adobe Reader to view these pages. If you do not have Adobe Reader, you can download a free copy at <http://www.adobe.com/products/acrobat/readstep2.html>. For more information on how to work with these pdfs, see our [pdf help page](#).

Each pdf has bookmarks that will easily navigate you to specific sections. Select the bookmark icon in the left panel of the pdf. Note that some of the pages contain photographs. Before printing photo pages, consider the extra time and printer ink required.

Printed copies
Printed copies of the Undergraduate Catalog are available on request from the Office of Undergraduate Admissions. Catalogs of the Graduate School and post-baccalaureate professional schools of the university are available on request from the dean of the appropriate school. Contact information for these offices is available on [People Finder](#).



<<Java Class>>

ReentrantReadWriteLock

- ReentrantReadWriteLock()
- ReentrantReadWriteLock(boolean)
- writeLock():WriteLock
- readLock():ReadLock
- isFair():boolean
- getReadLockCount():int
- isWriteLocked():boolean
- isWriteLockedByCurrentThread():boolean
- getWriteHoldCount():int
- getReadHoldCount():int
- hasQueuedThreads():boolean
- hasQueuedThread(Thread):boolean
- getQueueLength():int
- hasWaiters(Condition):boolean
- getWaitQueueLength(Condition):int
- toString()

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

Overview of Java Synchronizer Classes

- **ReentrantReadWriteLock**

- Improves performance when resources read more often than written
- Has many features
 - Both a blessing & a curse..



- **Reentrancy**

This lock allows both readers and writers to reacquire read or write locks in the style of a [ReentrantLock](#). Non-reentrant readers are not allowed until all write locks held by the writing thread have been released.

Additionally, a writer can acquire the read lock, but not vice-versa. Among other applications, reentrancy can be useful when write locks are held during calls or callbacks to methods that perform reads under read locks. If a reader tries to acquire the write lock it will never succeed.

- **Lock downgrading**

Reentrancy also allows downgrading from the write lock to a read lock, by acquiring the write lock, then the read lock and then releasing the write lock. However, upgrading from a read lock to the write lock is **not** possible.

- **Interruption of lock acquisition**

The read lock and write lock both support interruption during lock acquisition.

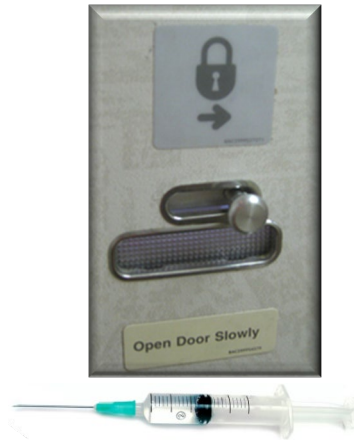
- **Condition support**

The write lock provides a [Condition](#) implementation that behaves in the same way, with respect to the write lock, as the [Condition](#) implementation provided by `newCondition()` does for [ReentrantLock](#). This [Condition](#) can, of course, only be used with the write lock.

The read lock does not support a [Condition](#) and `readLock().newCondition()` throws `UnsupportedOperationException`.

Overview of Java Synchronizer Classes

- **StampedLock**
 - A readers-writer lock that's more efficient than a `ReentrantReadWriteLock`



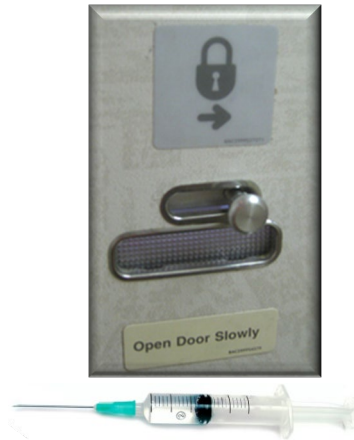
<<Java Class>>	
G StampedLock	
•	StampedLock()
•	writeLock():long
•	tryWriteLock():long
•	tryWriteLock(long, TimeUnit):long
•	writeLockInterruptibly():long
•	readLock():long
•	tryReadLock():long
•	tryReadLock(long, TimeUnit):long
•	readLockInterruptibly():long
•	tryOptimisticRead():long
•	validate(long):boolean
•	unlockWrite(long):void
•	unlockRead(long):void
•	unlock(long):void
•	tryConvertToWriteLock(long):long
•	tryConvertToReadLock(long):long
•	tryConvertToOptimisticRead(long):long
•	tryUnlockWrite():boolean
•	tryUnlockRead():boolean
•	isWriteLocked():boolean
•	isReadLocked():boolean
•	getReadLockCount():int
•	toString()
•	asReadLock():Lock
•	asWriteLock():Lock
•	asReadWriteLock():ReadWriteLock

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

Overview of Java Synchronizer Classes

- **StampedLock**

- A readers-writer lock that's more efficient than a `ReentrantReadWriteLock`
- Supports "optimistic" reads

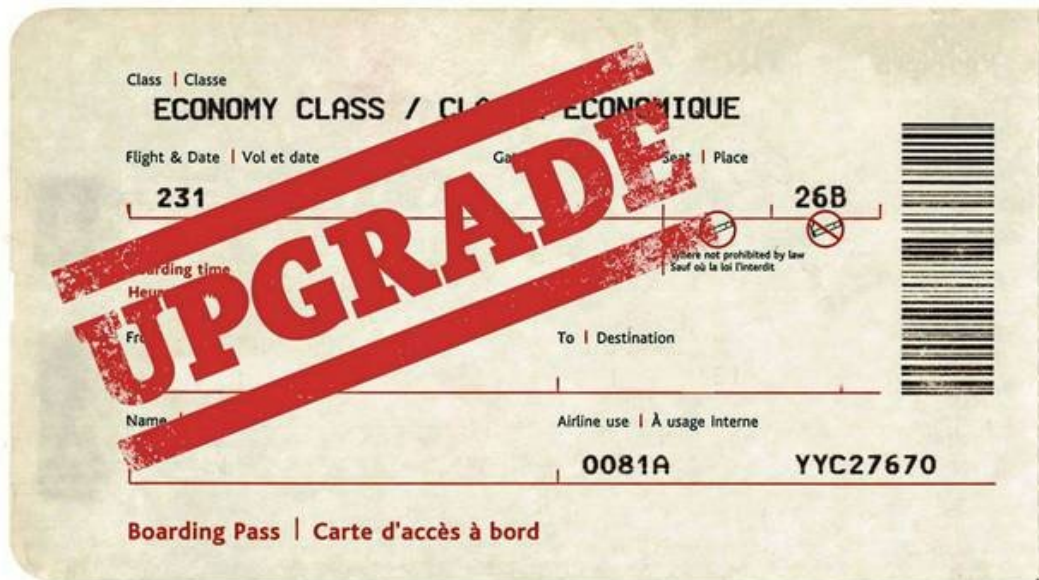


<<Java Class>>	
G StampedLock	
•	StampedLock()
•	writeLock():long
•	tryWriteLock():long
•	tryWriteLock(long,TimeUnit):long
•	writeLockInterruptibly():long
•	readLock():long
•	tryReadLock():long
•	tryReadLock(long,TimeUnit):long
•	readLockInterruptibly():long
•	tryOptimisticRead():long
•	validate(long):boolean
•	unlockWrite(long):void
•	unlockRead(long):void
•	unlock(long):void
•	tryConvertToWriteLock(long):long
•	tryConvertToReadLock(long):long
•	tryConvertToOptimisticRead(long):long
•	tryUnlockWrite():boolean
•	tryUnlockRead():boolean
•	isWriteLocked():boolean
•	isReadLocked():boolean
•	getReadLockCount():int
•	toString()
•	asReadLock():Lock
•	asWriteLock():Lock
•	asReadWriteLock():ReadWriteLock

Overview of Java Synchronizer Classes

- **StampedLock**

- A readers-writer lock that's more efficient than a ReentrantReadWriteLock
- Supports "optimistic" reads
- Also supports "lock upgrading"



<<Java Class>>	
G StampedLock	
•	StampedLock()
•	writeLock():long
•	tryWriteLock():long
•	tryWriteLock(long,TimeUnit):long
•	writeLockInterruptibly():long
•	readLock():long
•	tryReadLock():long
•	tryReadLock(long,TimeUnit):long
•	readLockInterruptibly():long
•	tryOptimisticRead():long
•	validate(long):boolean
•	unlockWrite(long):void
•	unlockRead(long):void
•	unlock(long):void
•	tryConvertToWriteLock(long):long
•	tryConvertToReadLock(long):long
•	tryConvertToOptimisticRead(long):long
•	tryUnlockWrite():boolean
•	tryUnlockRead():boolean
•	isWriteLocked():boolean
•	isReadLocked():boolean
•	getReadLockCount():int
•	toString()
•	asReadLock():Lock
•	asWriteLock():Lock
•	asReadWriteLock():ReadWriteLock


Overview of Java Synchronizer Classes



















- **Semaphore**

- Maintains permits that control thread access to limited # of shared resources



<<Java Class>>

 **Semaphore**

-  Semaphore(int)
-  Semaphore(int,boolean)
-  acquire():void
-  acquireUninterruptibly():void
-  tryAcquire():boolean
-  tryAcquire(long, TimeUnit):boolean
-  release():void
-  acquire(int):void
-  acquireUninterruptibly(int):void
-  tryAcquire(int):boolean
-  tryAcquire(int, long, TimeUnit):boolean
-  release(int):void
-  availablePermits():int
-  drainPermits():int
-  isFair():boolean
-  hasQueuedThreads():boolean
-  getQueueLength():int
-  toString()

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

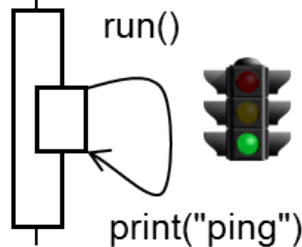
Overview of Java Synchronizer Classes

• Semaphore

- Maintains permits that control thread access to limited # of shared resources
- Operations need not be fully bracketed..



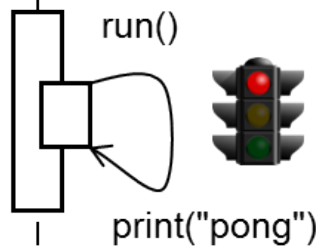
ping : ➤
PingPongThread



1


Semaphores

pong : ➤
PingPongThread



0

<<Java Class>>

 **Semaphore**

- Semaphore(int)
- Semaphore(int,boolean)
- acquire():void
- acquireUninterruptibly():void
- tryAcquire():boolean
- tryAcquire(long, TimeUnit):boolean
- release():void
- acquire(int):void
- acquireUninterruptibly(int):void
- tryAcquire(int):boolean
- tryAcquire(int, long, TimeUnit):boolean
- release(int):void
- availablePermits():int
- drainPermits():int
- isFair():boolean
- hasQueuedThreads():boolean
- getQueueLength():int
- toString()

Overview of Java Synchronizer Classes









- **ConditionObject**

- Allows a thread to wait until some condition become true



<<Java Class>>

ConditionObject

-  ConditionObject()
-  signal():void
-  signalAll():void
-  awaitUninterruptibly():void
-  await():void
-  awaitNanos(long):long
-  awaitUntil(Date):boolean
-  await(long, TimeUnit):boolean

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html

Overview of Java Synchronizer Classes

- **ConditionObject**

- Allows a thread to wait until some condition become true
- Always used in conjunction with a ReentrantLock



<<Java Class>>	
G ReentrantLock	
● ^C	ReentrantLock()
● ^C	ReentrantLock(boolean)
●	lock():void
●	lockInterruptibly():void
●	tryLock():boolean
●	tryLock(long,TimeUnit):boolean
●	unlock():void
●	newCondition():Condition

<<Java Class>>	
G ConditionObject	
● ^C	ConditionObject()
● ^F	signal():void
● ^F	signalAll():void
● ^F	awaitUninterruptibly():void
● ^F	await():void
● ^F	awaitNanos(long):long
● ^F	awaitUntil(Date):boolean
● ^F	await(long,TimeUnit):boolean

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/AbstractQueuedSynchronizer.ConditionObject.html

Overview of Java Synchronizer Classes

- **CountDownLatch**

- Allows one or more threads to wait on the completion of operations in other threads

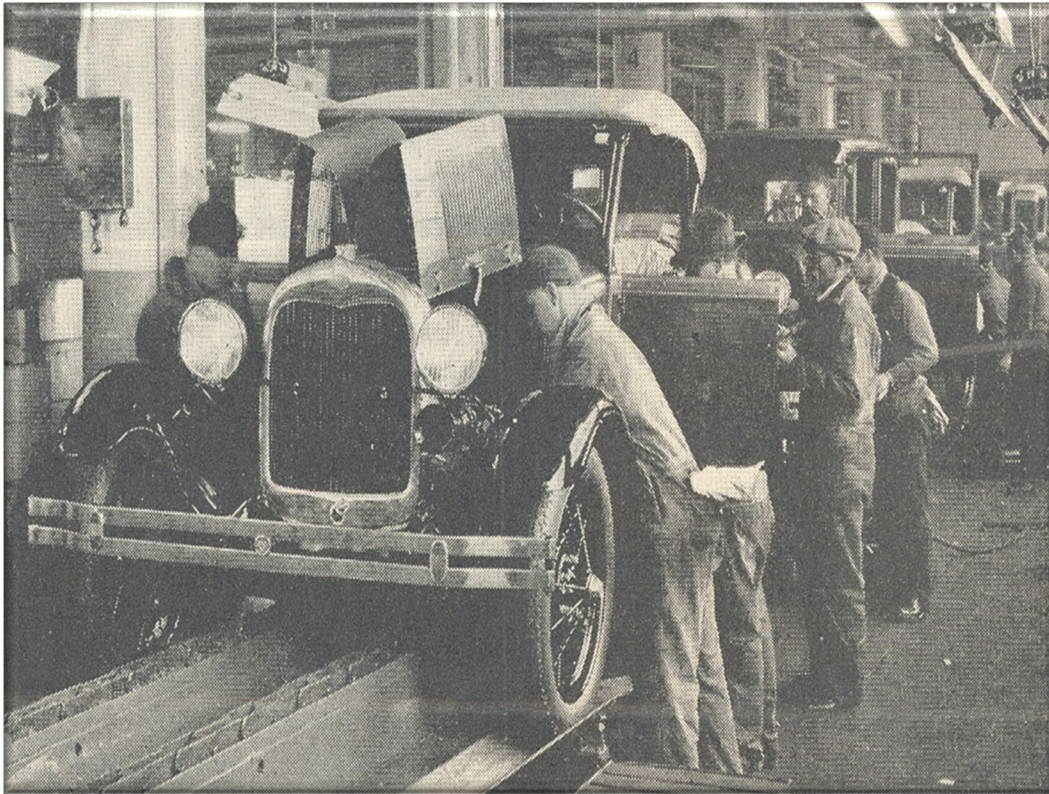


<<Java Class>>	
G CountDownLatch	
•	CountDownLatch(int)
•	await():void
•	await(long, TimeUnit):boolean
•	countDown():void
•	getCount():long
•	toString()



Overview of Java Synchronizer Classes

- **CyclicBarrier**
 - Allows a set of threads to all wait for each other to reach a common barrier point



<<Java Class>>	
G CyclicBarrier	
•	CyclicBarrier(int,Runnable)
•	CyclicBarrier(int)
•	getParties():int
•	await():int
•	await(long,TimeUnit):int
•	isBroken():boolean
•	reset():void
•	getNumberWaiting():int



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

Overview of Java Synchronizer Classes

- **Phaser**

- A more flexible, reusable, & dynamic barrier synchronizer that subsumes CyclicBarrier & CountdownLatch



<<Java Class>>

Phaser

- **Phaser()**
- **Phaser(int)**
- **Phaser(Phaser)**
- **Phaser(Phaser,int)**
- **register():int**
- **bulkRegister(int):int**
- **arrive():int**
- **arriveAndDeregister():int**
- **arriveAndAwaitAdvance():int**
- **awaitAdvance(int):int**
- **awaitAdvanceInterruptibly(int):int**
- **awaitAdvanceInterruptibly(int,long,TimeUnit):int**
- **forceTermination():void**
- **getPhase():int**
- **getRegisteredParties():int**
- **getArrivedParties():int**
- **getUnarrivedParties():int**
- **getParent():Phaser**
- **getRoot():Phaser**
- **isTerminated():boolean**
- **onAdvance(int,int):boolean**
- **toString()**

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

End of Synopsis of Java Synchronizer Classes