

Processes \_\_\_\_\_ than threads.

You are correct!

require more overhead to create

Threads that belong to the same process share that process's address space.

Thread.getState()

This can return : NEW, Runnable, Time\_wait, and Terminated

CurrentThread()

Returns the current thread that is running

**Thread ID**

Given a positive long number

getID()

**Thread Name**

String used for Identification

Multiple threads can have the same name but not the same ID

* You can change the name of a thread with the setName(Sting name) function
* And get the thread name with getName()

**Thread Priority**

* 1 = lowest
* 10 = highest
* Void setPriority(int newPriority)
* Int getPriority()

The Threads do not know who their parent is!!!!

**Multiple ways to create a thread**

* Extend the Thread Class
* Implement Runnable
  + Class that will be executed by a thread
  + Requires a run() function

**Differences**

* You cannot extend the Thread class
* Runnable allows for multiple extensions
* Each instance of Thread is a separate object
* Multiple threads can share a single Runnable object (reduce memory usage)

Due to these items stated above the Runnable class is used more often

**Garbage Collector**

* If you create a garbage collection thread, then it needs to be a daemon thread
  + This thread will not affect the exiting of a program
  + These threads are detached from the main system
  + They will however be terminated non gracefully which means that the thread could cause corruption if it is writing to a file so make sure that these thread do not have any functions that can cause corruption if closed prematurely
* You can change a thread to being daemon by using
  + threadName.setDaemon(true);
  + daemon threads need to be set to daemon before starting!!!

**Data Race:**

Import java.util.concurrent.atomic.\*

Static AtomicInteger garlicCount = new AtomicInteger(initialValue: 0);

Public run() {

For int I = 0 I < 10 ; i++

garlicCount.incrementAndGet();

you can use this to share a counter between threads

**Synchronization:**

Private static synchronized void addGarlic()

{  
garlicCount++;

}

This means that a lock is created for this method automatically !!!

Static is required to make sure that the threads are using the same lock!!!

Shopper.class can be used as the synchronized object to make it work as anything that uses the class

**Reentrant lock**

* can be locked multiple times by the same thread
* needs to be unlocked an equal amount of times
* these can be used in recursion

**Try lock**

* non-blocking lock
* if the mutex is already in use then return false and continue doing your work
* if the mutex is available then it returns TRUE and can then take the key

**Reader-Writer-Lock**

* you have 2 different lock types
* a reading mode lock will allow multiple threads to read-only the document
* the write-only lock will only allow 1 thread to write to the document.

**Lock Order**

* make sure that all locks are acquired in the same order by all threads!!!
* This will prevent DEADLOCKS

**LOCK Timeout**

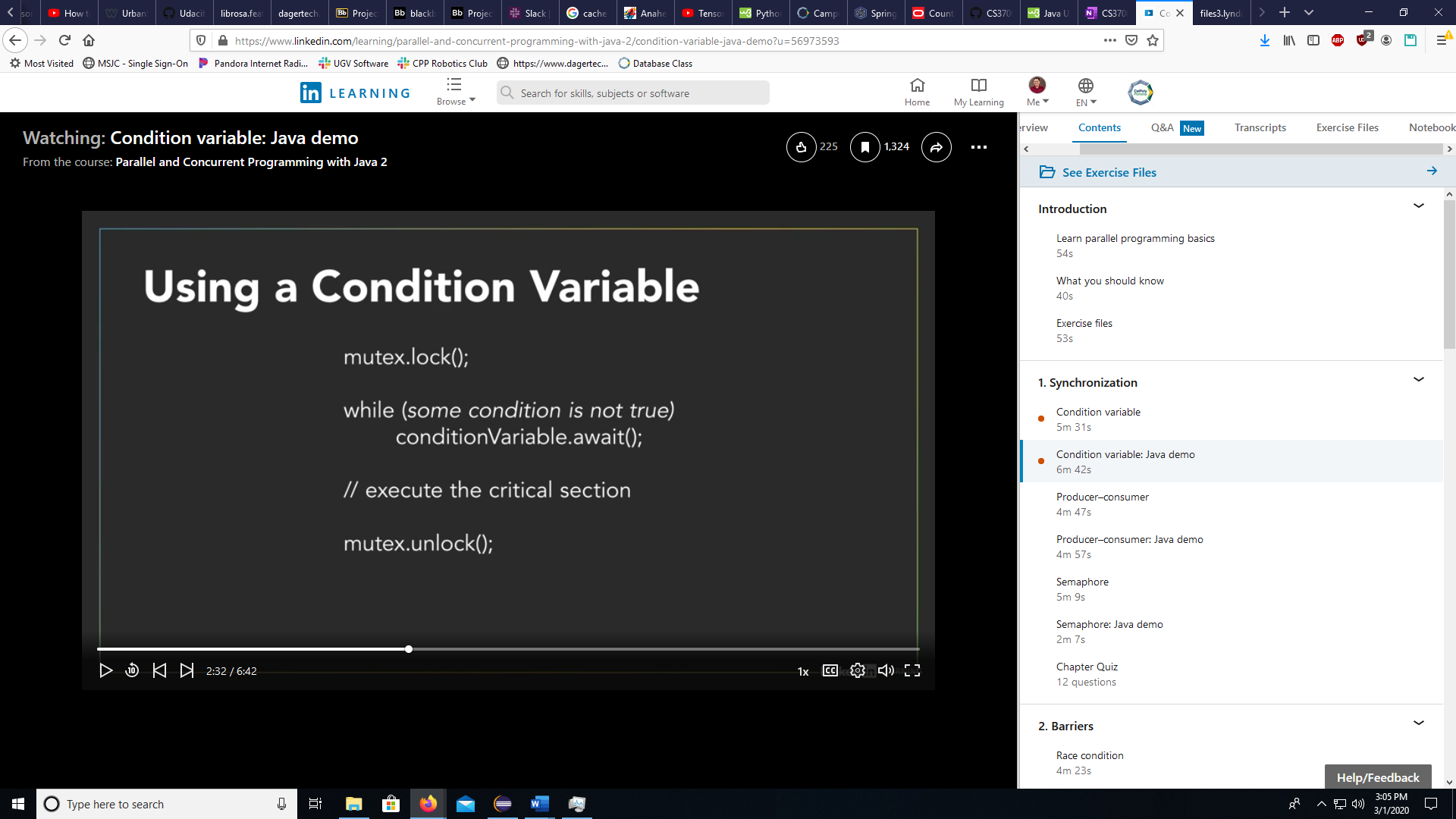
* Acquire a lock for a set amount of time until it needs to put the lock back if the other lock is not available

**Lock Abandonment**

* To make it so this doesn’t happen we can write a try and a finally system
* In the finally clause you want to put the release of the lock in it. This way the thread that hits an exception then will for sure release the lock.

**Starvation**

* A thread that can never gain the data it needs because a different thread always has the lock



In addition to modifying the counter value, what else does calling the semaphore's release() method do?

You are correct!

Signal another thread waiting to acquire the semaphore.

What does the semaphore's release() method do to the counter value?

You are correct!

Always increment the counter's value.

What does the semaphore's acquire() method do to the counter value?

You are correct!

If the counter is positive, decrement its value.

What is the difference between a binary semaphore and a mutex?

You are correct!

The binary semaphore can be acquired and released by different threads.

What happens if the producer puts elements into a fixed-length queue faster than the consumer removes them?

You are correct!

The queue will fill up and cause an error.

Which architecture consists of a chained-together series of producer-consumer pairs?

You are correct!

pipeline

How should the average rates of production and consumption be related in a producer-consumer architecture?

You are correct!

The consumption rate should be greater than or equal to the production rate.

When should a thread typically signal a condition variable?

You are correct!

after doing something to change the state associated with the condition variable but before unlocking the associated mutex

Why would you use the condition variable's signal() method instead of signalAll()?

You are correct!

You only need to wake up one waiting thread and it does not matter which one.

Condition variables serve as a holding place for threads to wait for a certain condition before continuing execution .

Condition variables work together with which other mechanism serving as a monitor?

You are correct!

a mutex

**Barriers**: