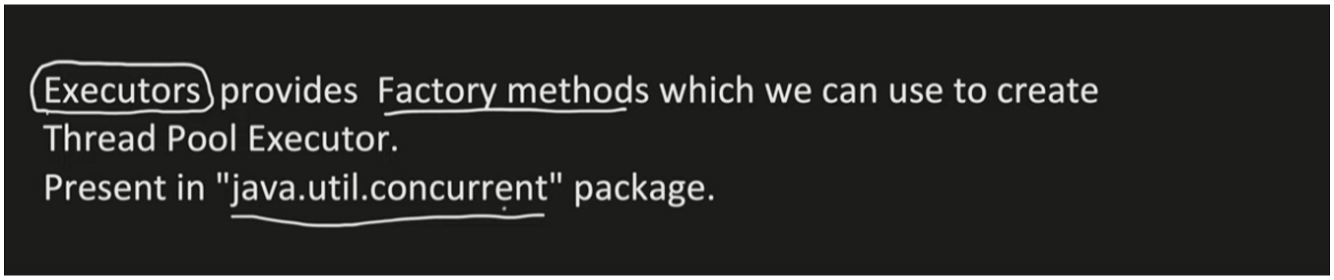
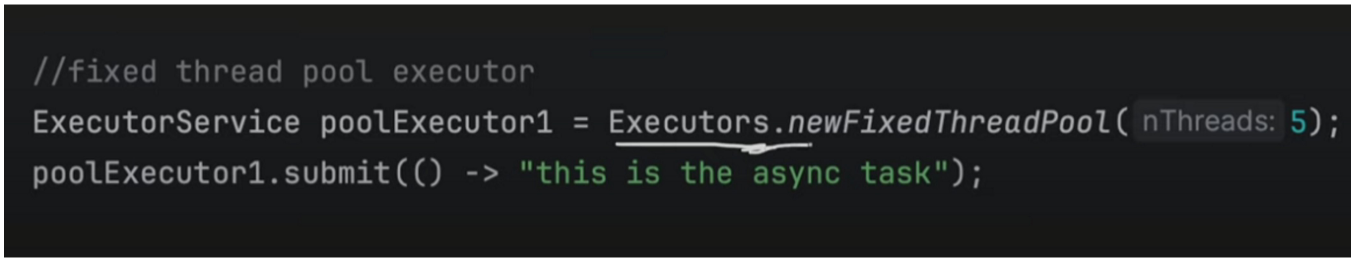
Lesson 8 : ExecutorUtility and Fork-Join Pool

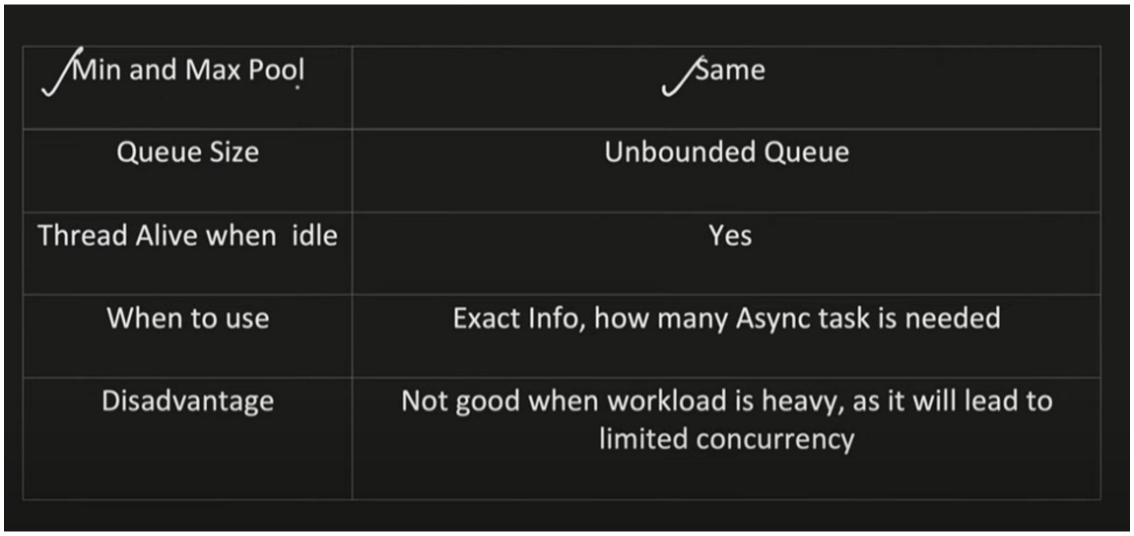
Tuesday, May 14, 2024 11:15 PM


Executors provides Factory methods which we can use to create Thread Pool Executor.
Present in "java.util.concurrent" package.

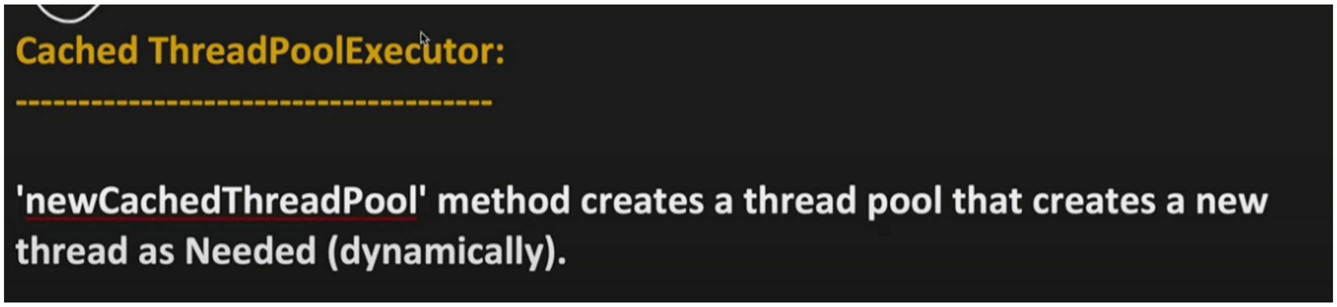
Fixed ThreadPoolExecutor

'newFixedThreadPool' method creates a thread pool executor with a fixed no of threads.

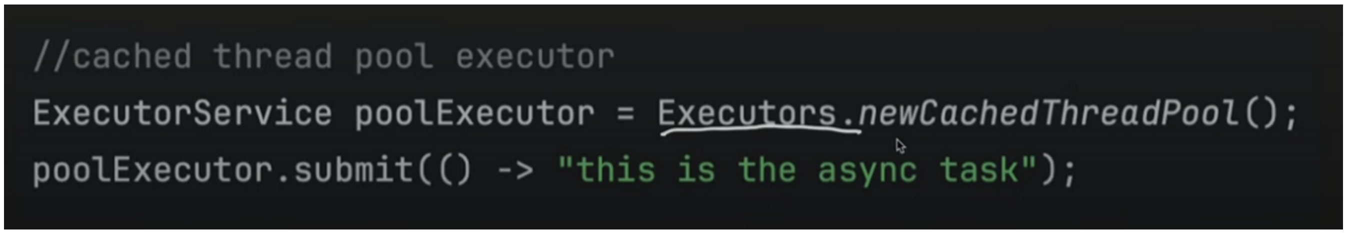

```
//fixed thread pool executor
ExecutorService poolExecutor1 = Executors.newFixedThreadPool( nThreads: 5);
poolExecutor1.submit(() -> "this is the async task");
```



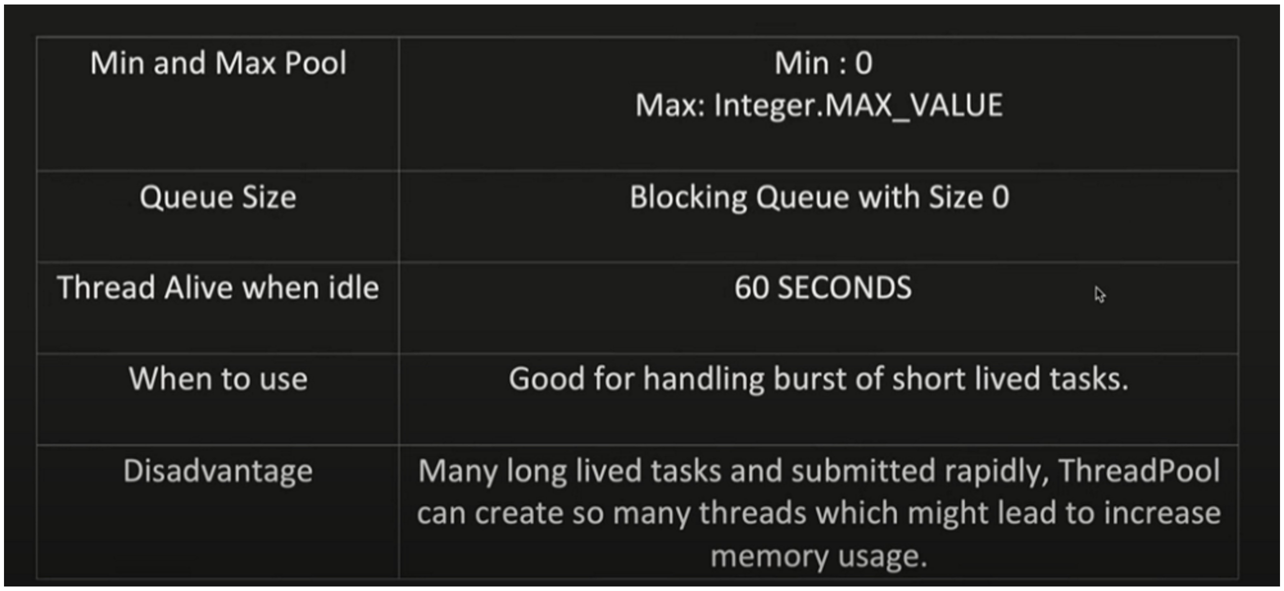
✓Min and Max Pool	✓Same
Queue Size	Unbounded Queue
Thread Alive when idle	Yes
When to use	Exact Info, how many Async task is needed
Disadvantage	Not good when workload is heavy, as it will lead to limited concurrency


Cached ThreadPoolExecutor:

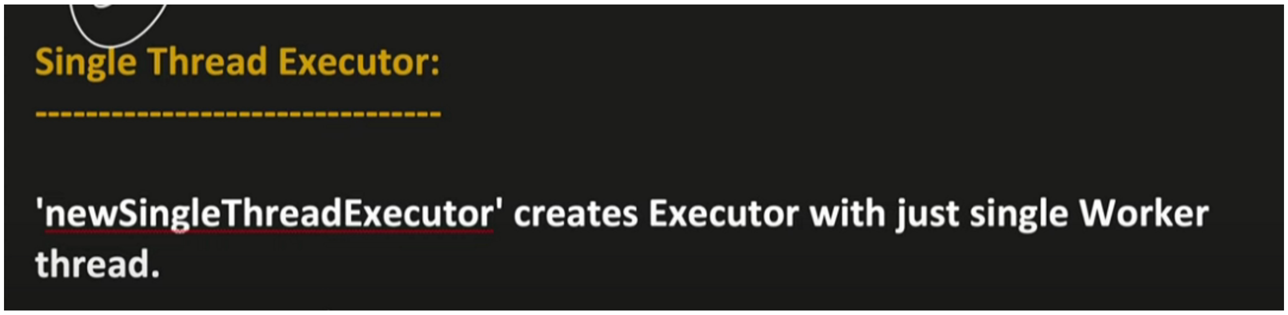
'newCachedThreadPool' method creates a thread pool that creates a new thread as Needed (dynamically).



```
//cached thread pool executor
ExecutorService poolExecutor = Executors.newCachedThreadPool();
poolExecutor.submit(() -> "this is the async task");
```

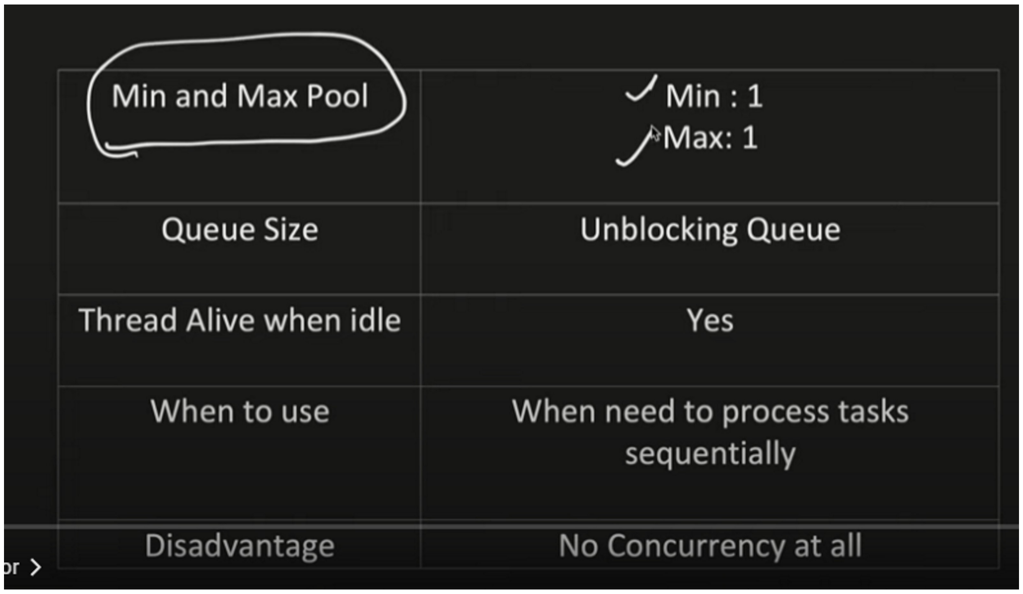


Min and Max Pool	Min : 0 Max: Integer.MAX_VALUE
Queue Size	Blocking Queue with Size 0
Thread Alive when idle	60 SECONDS
When to use	Good for handling burst of short lived tasks.
Disadvantage	Many long lived tasks and submitted rapidly, ThreadPool can create so many threads which might lead to increase memory usage.



Single Thread Executor:

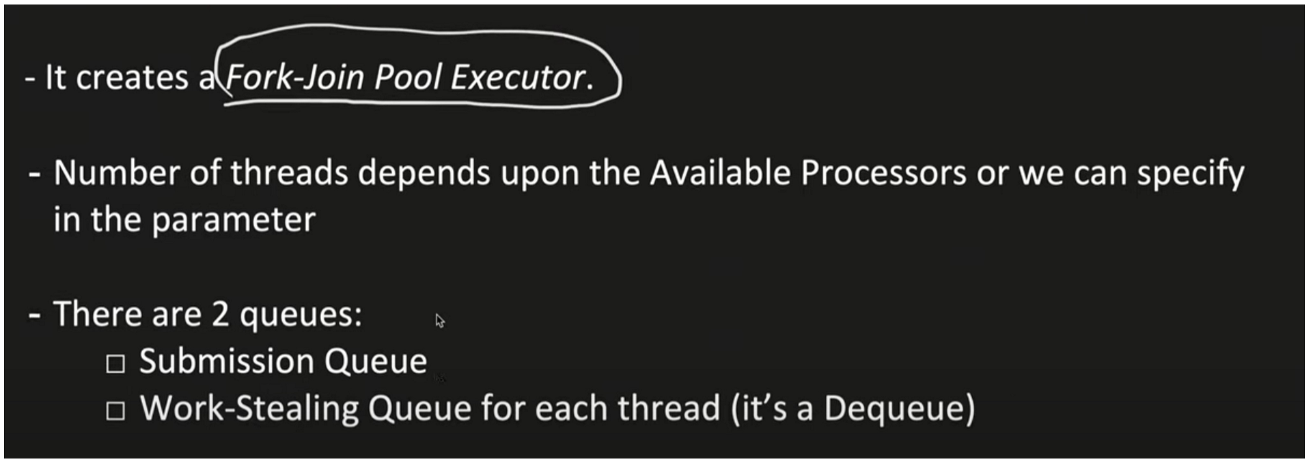
'newSingleThreadExecutor' creates Executor with just single Worker thread.



Min and Max Pool	✓ Min : 1 ✓ Max: 1
Queue Size	Unblocking Queue
Thread Alive when idle	Yes
When to use	When need to process tasks sequentially
Disadvantage	No Concurrency at all

WorkStealing Pool Executor :

Before knowing about workStealing pool executor, we need to know about Fork-Join Pool Executor



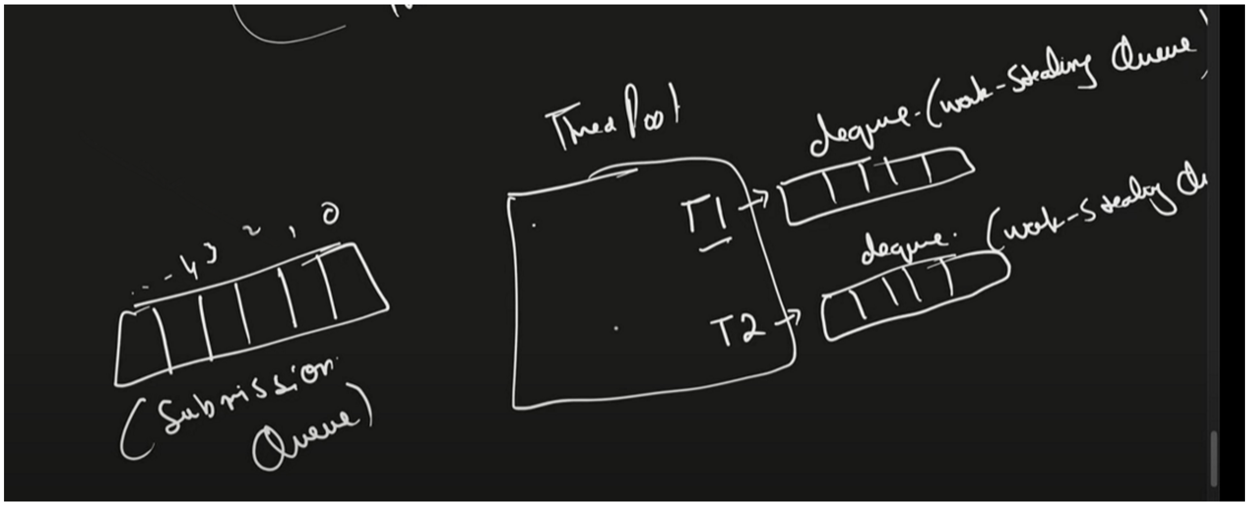
- It creates a *Fork-Join Pool Executor*.
- Number of threads depends upon the Available Processors or we can specify in the parameter
- There are 2 queues:
  - □ Submission Queue
  - □ Work-Stealing Queue for each thread (it's a Dequeue)

What is Fork-Join

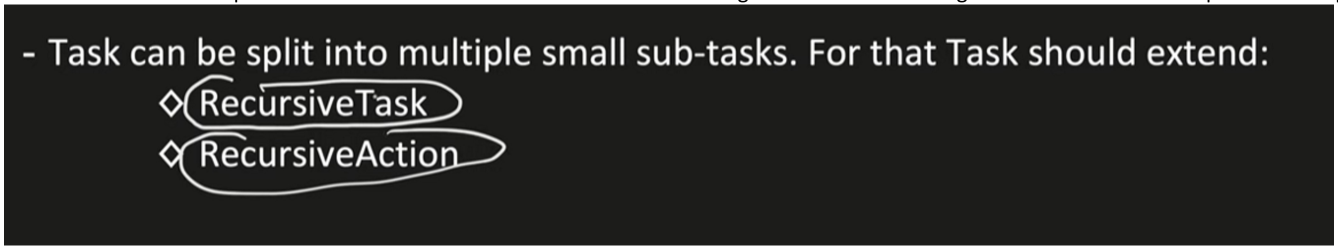
Big task is broken into smaller tasks and assigned to different thread using fork(). The subtask can be further divided also using fork() and ultimately all these pieces are stitched together using join(). This ensures parallelism.

We know the usual workflow that there is a thread Pool where fixed number of threads are ready to pickup tasks. There is also a queue which has tasks queued for thread pool to pick when they get free. Work Stealing pool has extended this concept. How ?

Each thread has its own dequeue (double ended queue) also, that is known as work stealing queue.



Tasks which cannot be split into smaller tasks work as usual. But work stealing feature can be leveraged for tasks that can be split into multiple sub-tasks.

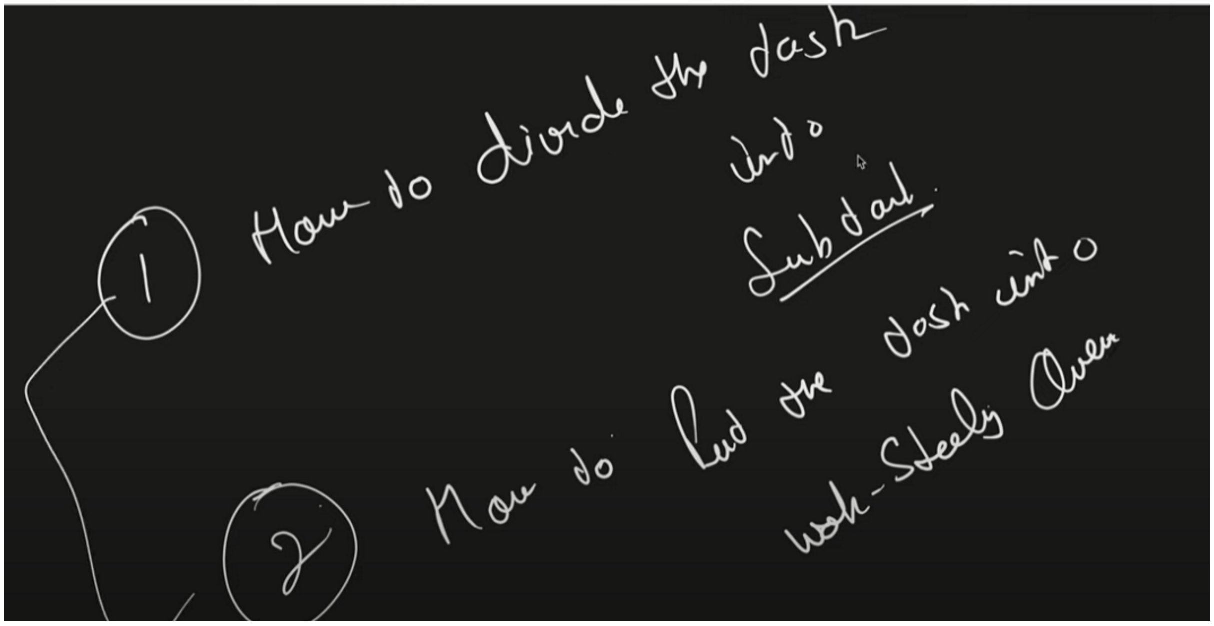


- Task can be split into multiple small sub-tasks. For that Task should extend:
  - ◇ RecursiveTask
  - ◇ RecursiveAction

The split tasks are queued in work-stealing queue of a thread. So when the other thread is done with its tasks, it picks up tasks as following priority :

1. Own work stealing queue.
2. Submission queue
3. Other thread's work stealing queue.

That's why the name work-stealing as it is stealing the task from other thread's dequeue if unoccupied and submission queue also empty.
It is dequeue because original thread picks up task from front but other thread that is stealing, will take task from back of the queue.



1 How to divide the task into Subtask.

2 How to put the task into work-Stealing Queue

Dividing the task.
We can use two methods, recursive task and recursive action.
Whenever the task needs to return something, then recursive task method is used otherwise recursive action method.

See coding example how tasks are divided and submitted to work-stealing queue.