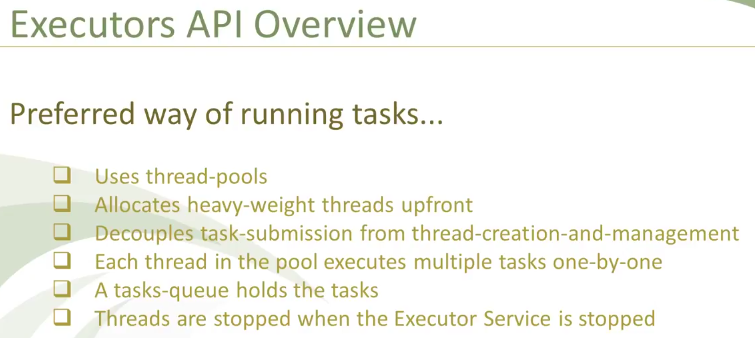
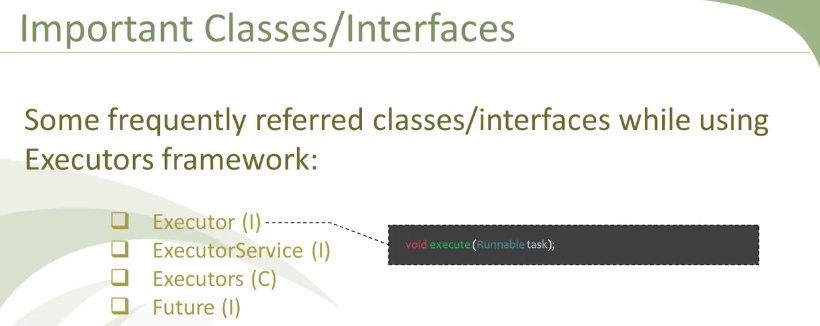


**Executor service:**

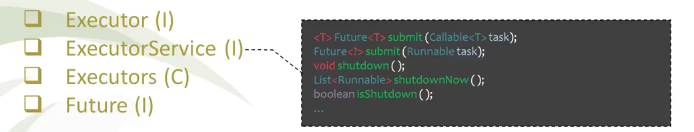
Write code for defining the task to run it in different task. We should provide this task to the executor service. Executor Service automatically creates thread and

run our code in that thread, without having us to create the thread. Thread creation has CPU overhead. It uses thread-pool internally. It is recycled and reused.



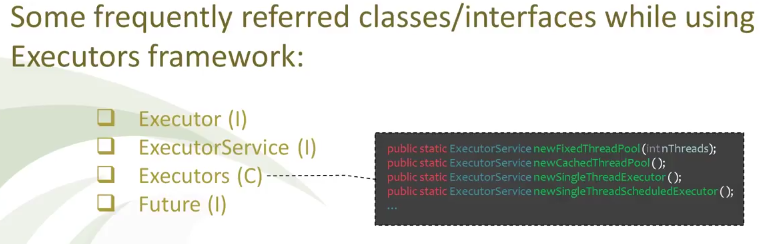


Executor decouples the task submission step from thread creation and running steps.



It provides life cycle. Initialization service and destruction phase. We call a task by callable interface when we want that task to return the value. Runnable does not return anything.

Using shutdown, we shut the task. No new other task can be submitted.

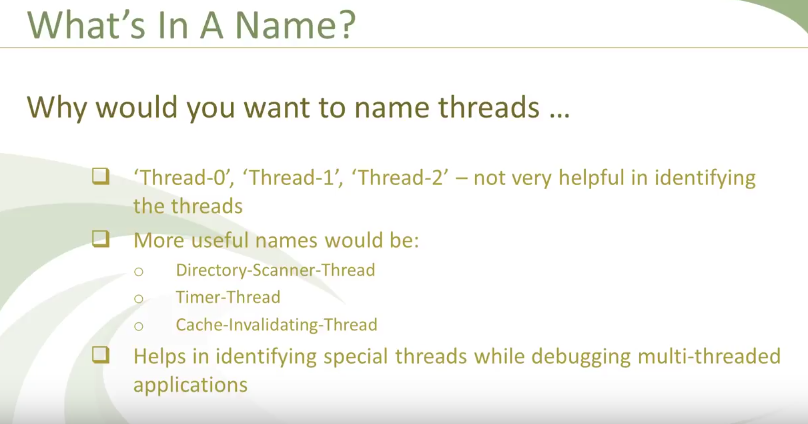


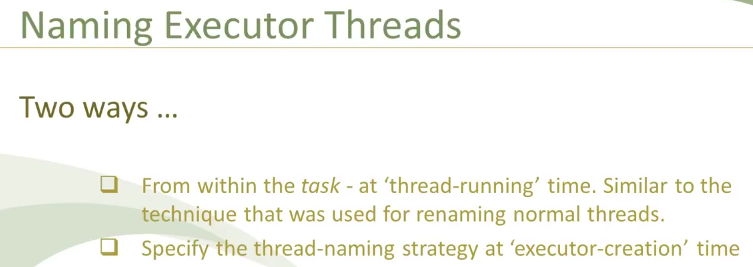
Factory class that constructs and returns various kind of instances of executor service with commonly useful configuration settings.

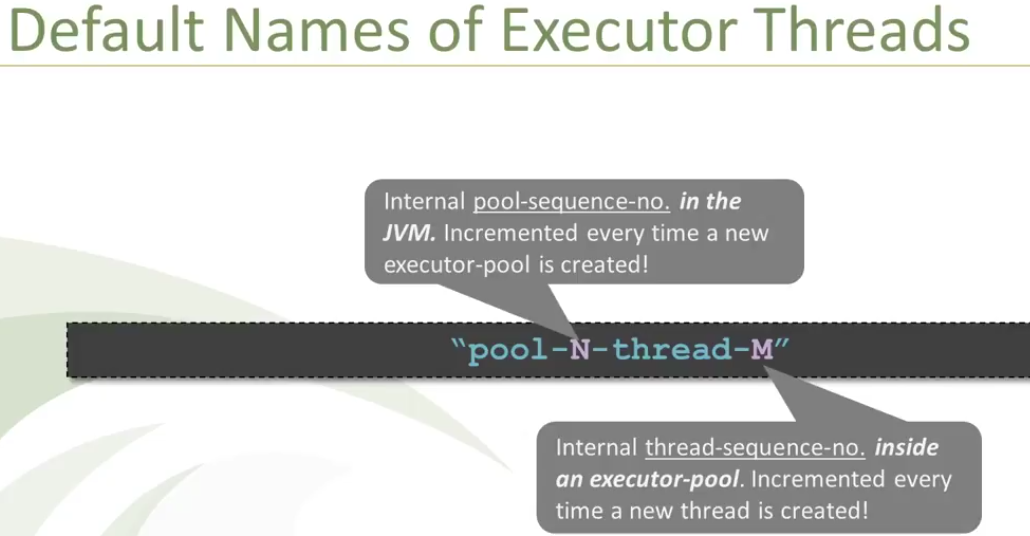
1. Public static ExecutorService newFixedThreadPool (int thead)-🡪 Only n number of threads can be executed
2. Public static ExecutorService newCachedThreadPool ();
3. Public static ExecutorService newSingleThreadExecutor (); ---One by One from queue. But can never execute more than one task simultaneously
4. Public static ExecutorService newSingleThreadScheduledExecutor ();-🡪Whenever we submit a callable for execution the ExecutorService returns an instance of Future. We can use get method of future to check the status of task.













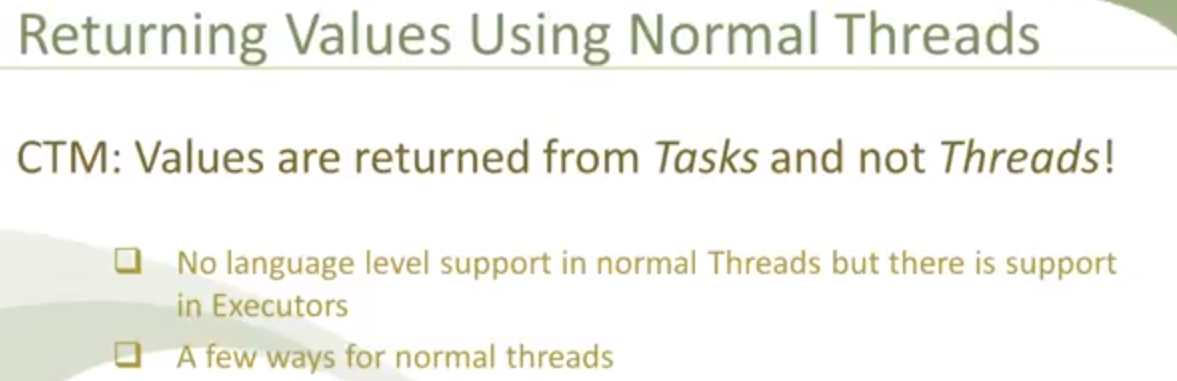
In Fixed Thread-Pool executor the number of thread is fixed. If the number of submitted task exceeds the number of threads created in the fixed thread pool. The extra thread will have to wait in the thread queue till the time some thread finishes its execution and becomes available to execute another task from queue.

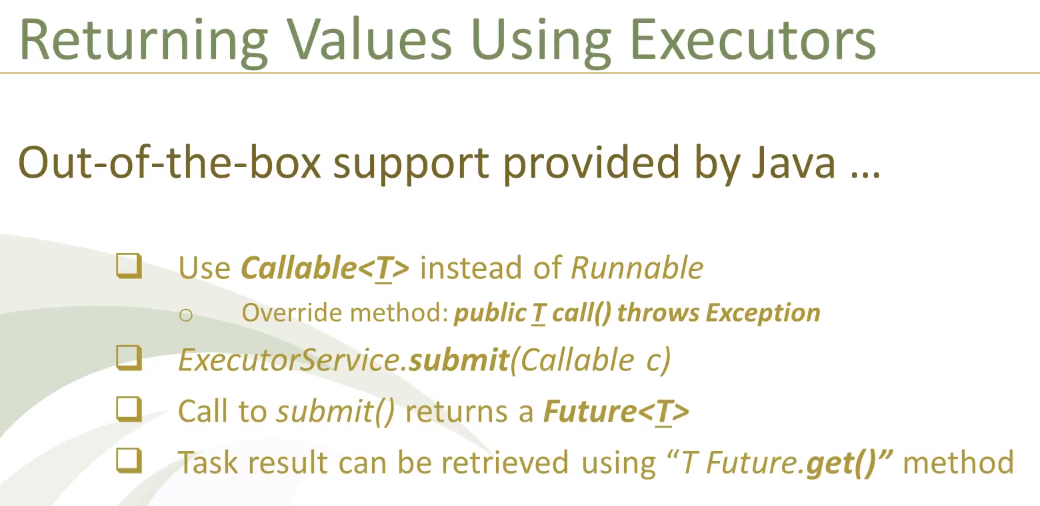
In cached Thread Pool as many number of threads are created as the number of threads submitted to execute concurrently. So that no task will have to wait for execution. No concept of wait queue.

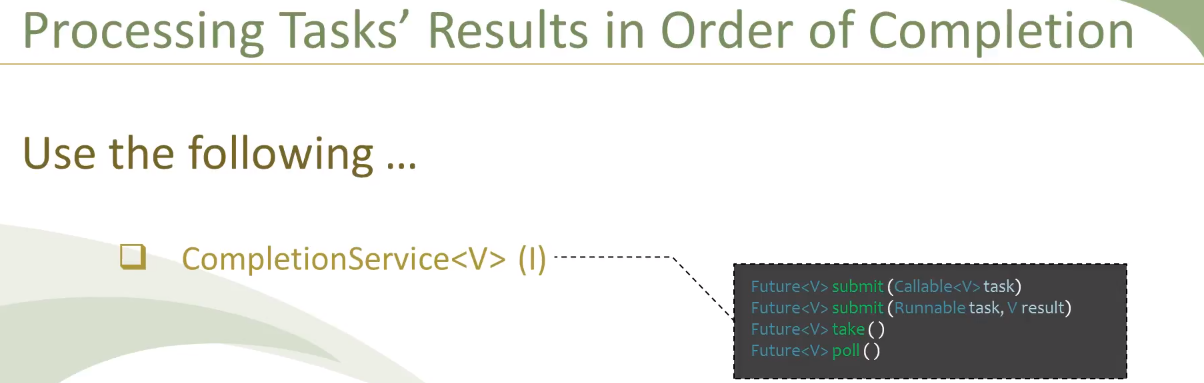
**How to return values from Normal thread**

**1st Way: **

**2nd Way: **



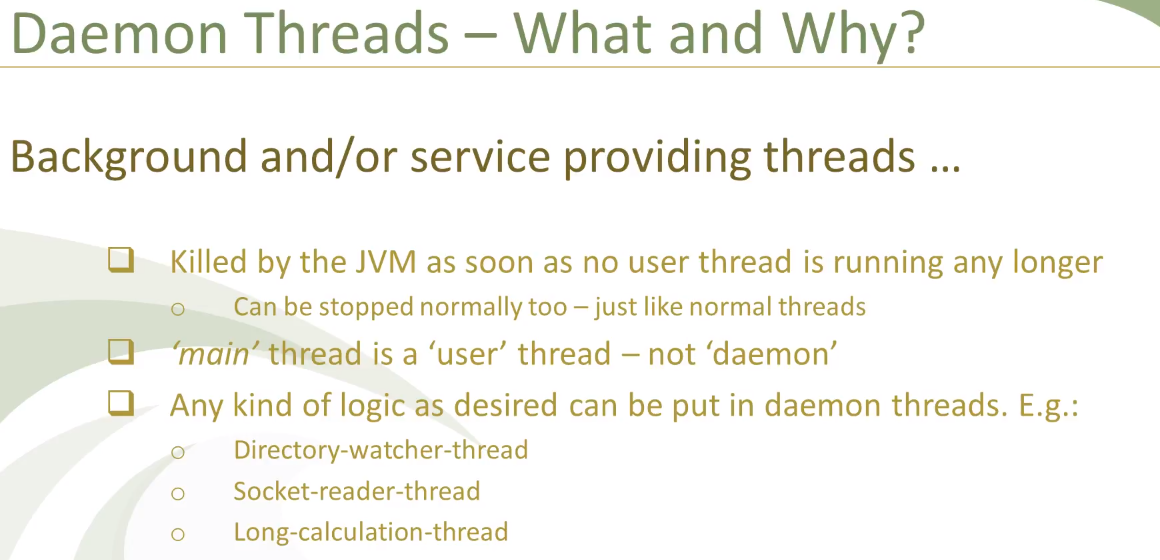




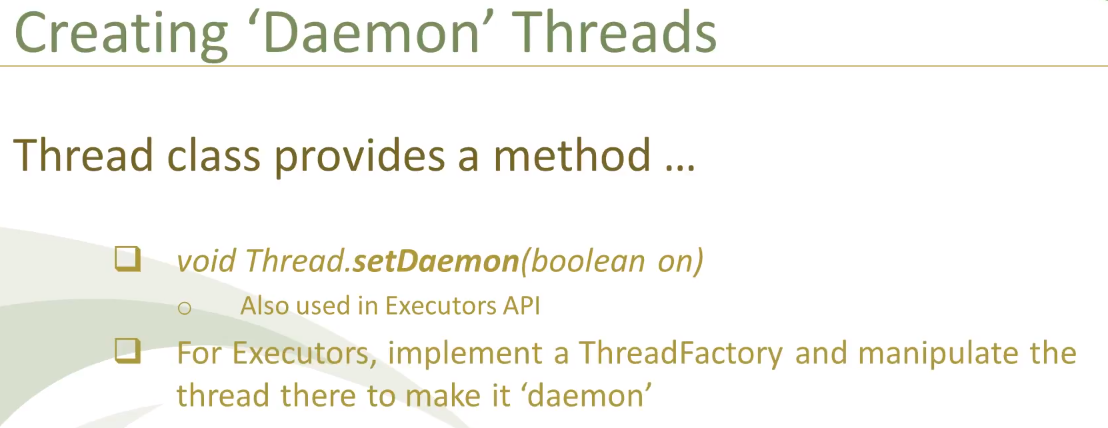
JDK provides concrete implementation class:

**Daemon Threads:**

Special kind of threads that keep on running in the background and may also provide services to user threads. Its existence depends upon the normal user threads. If one user thread is running then Daemon threads keep on running.

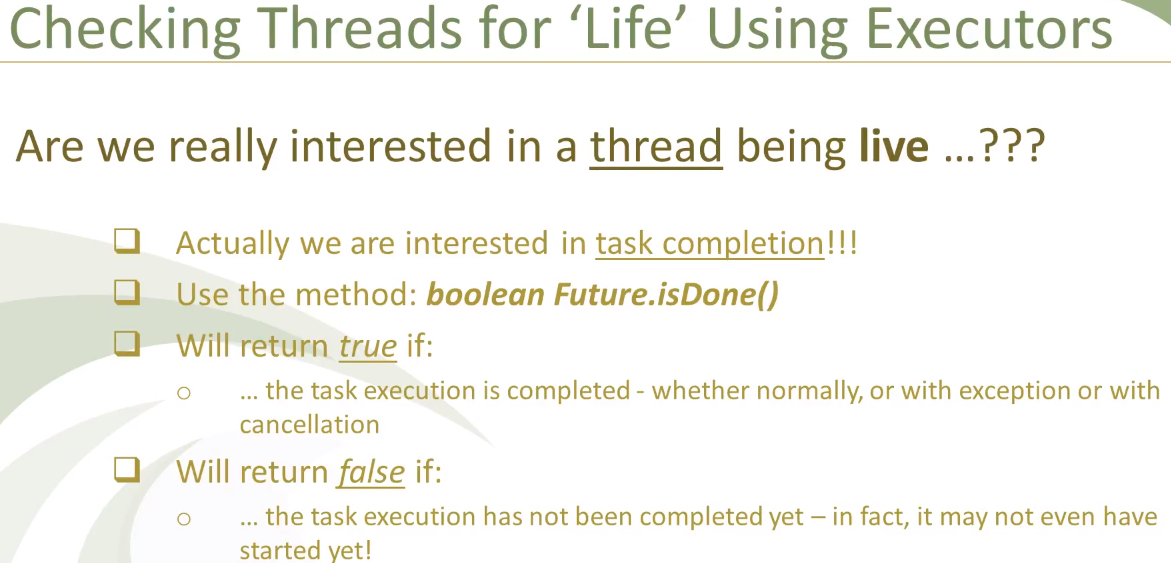


Most of the time daemon thread keeps on running till the application is down. But it may not be true always.



Executor thread do not terminate or die after finishing a task. Rather they go into wait state if no more task is there to be executed.





Volatile variable will get the most recently changed value by shared thread



