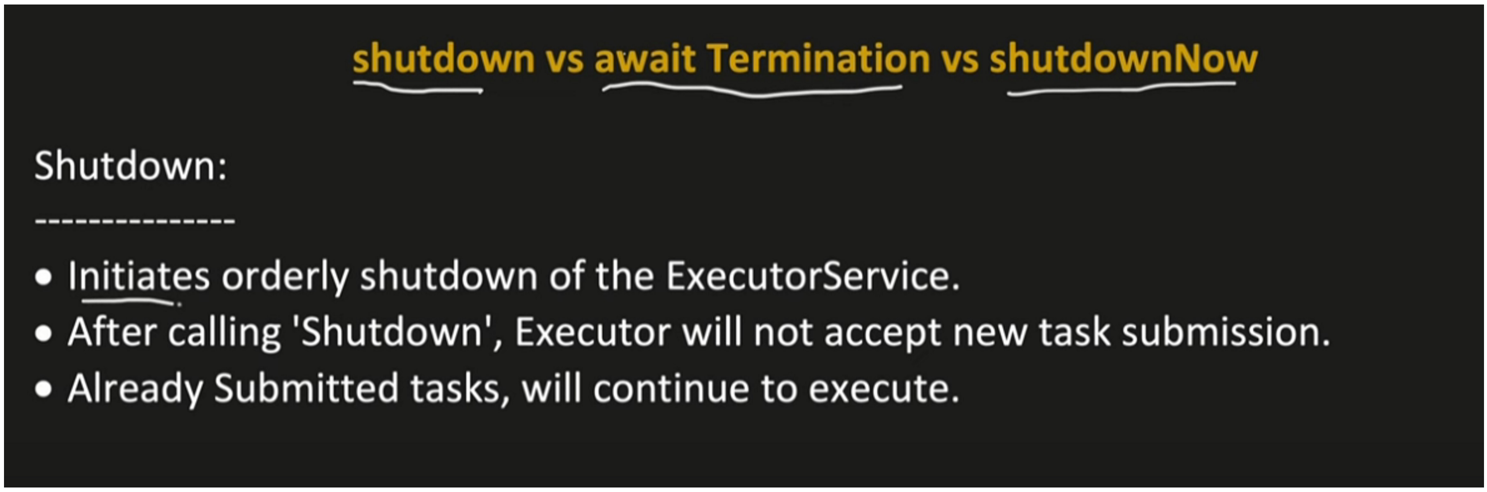
Lesson 9 : ShutdownVsShutdownNow

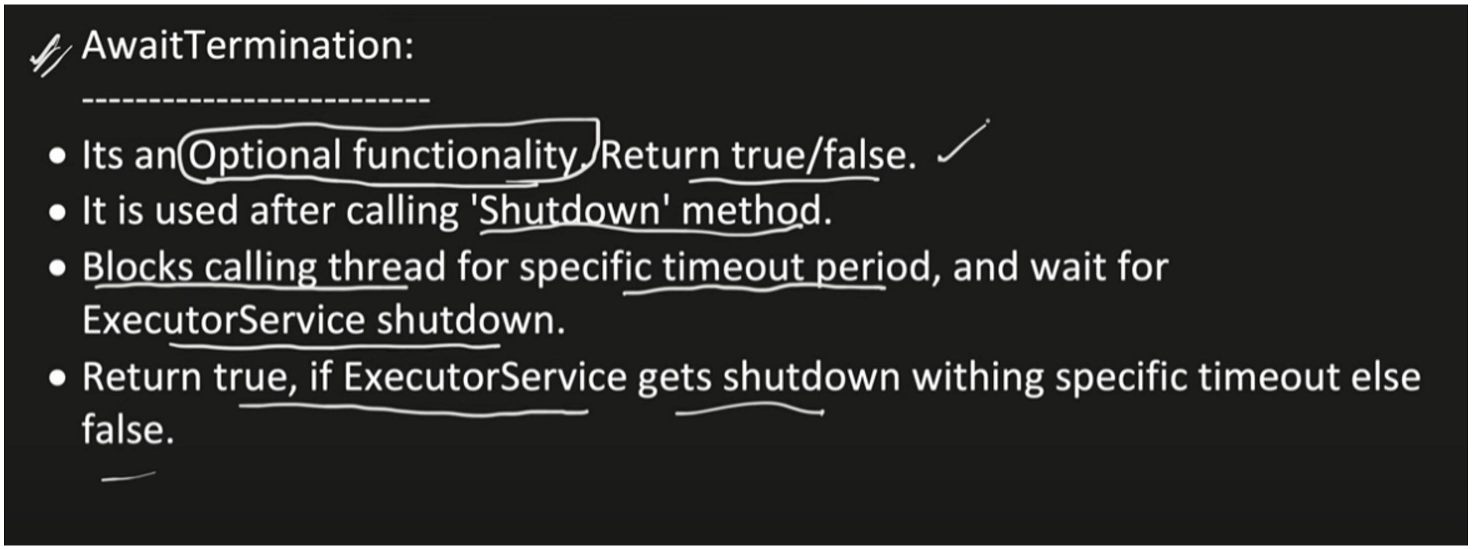
Monday, May 20, 2024 10:16 PM



shutdown vs await Termination vs shutdownNow

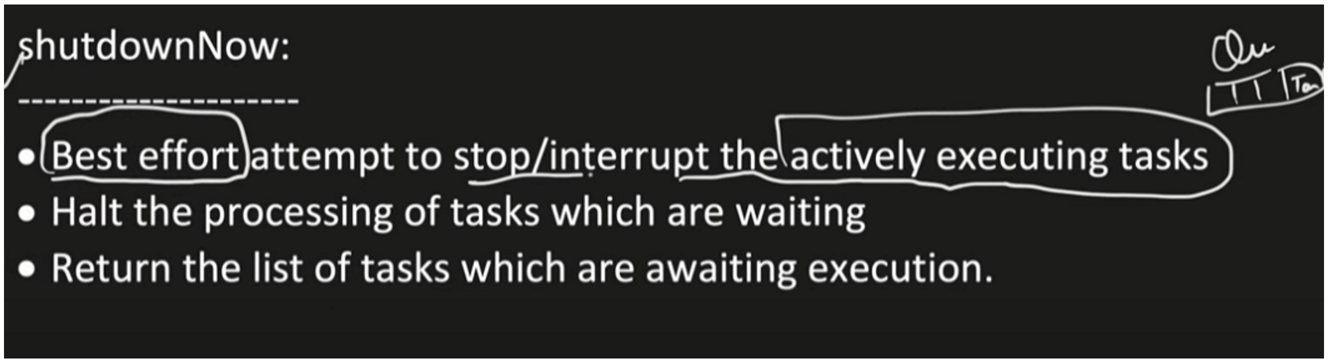
Shutdown:

- Initiates orderly shutdown of the ExecutorService.
- After calling 'Shutdown', Executor will not accept new task submission.
- Already Submitted tasks, will continue to execute.



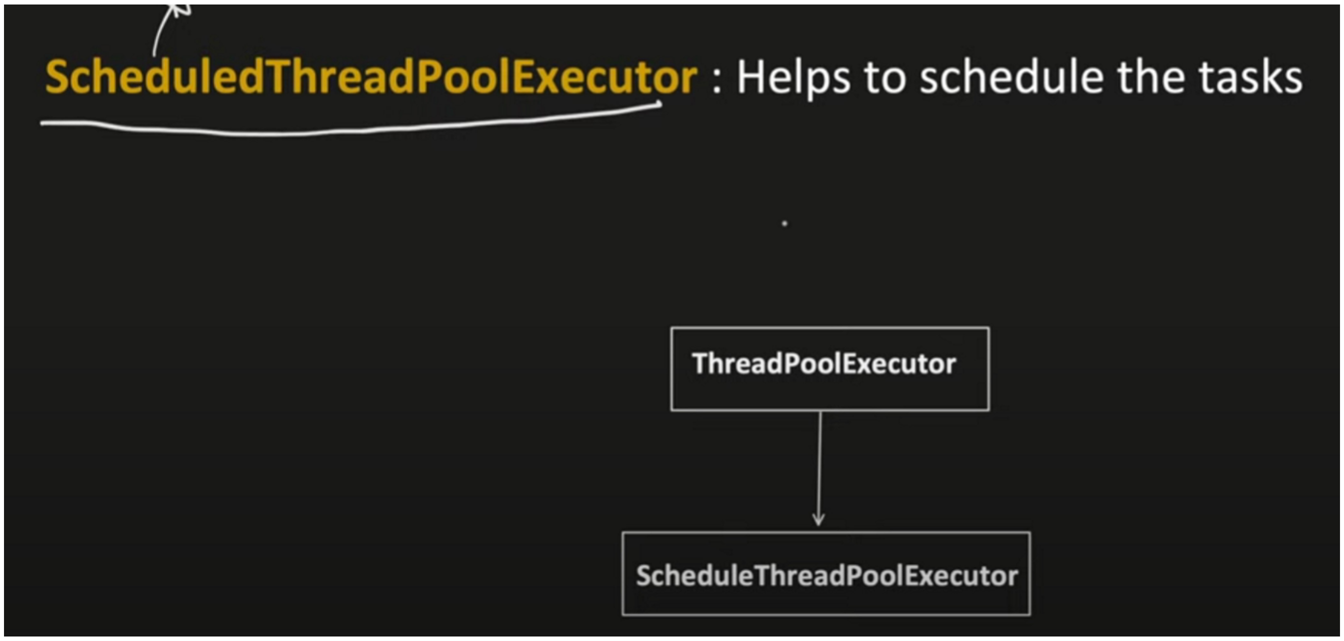
AwaitTermination:

- Its an Optional functionality. Return true/false.
- It is used after calling 'Shutdown' method.
- Blocks calling thread for specific timeout period, and wait for ExecutorService shutdown.
- Return true, if ExecutorService gets shutdown withing specific timeout else false.

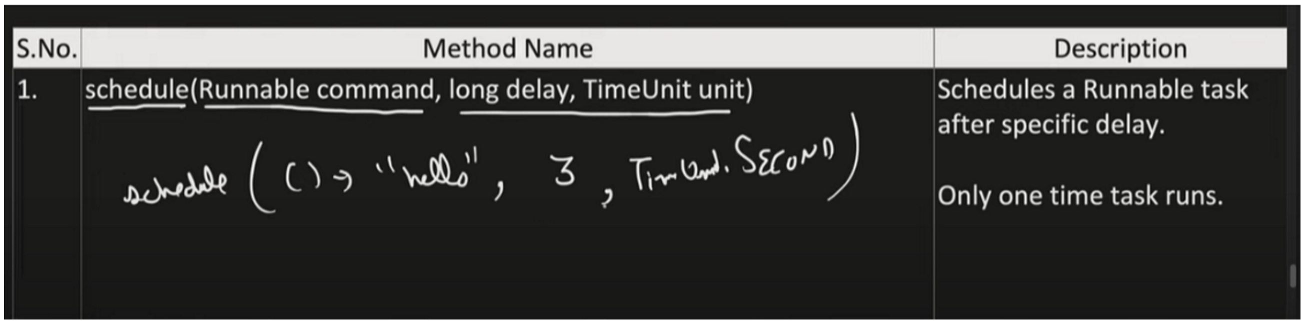


shutdownNow:

- Best effort attempt to stop/interrupt the actively executing tasks
- Halt the processing of tasks which are waiting
- Return the list of tasks which are awaiting execution.

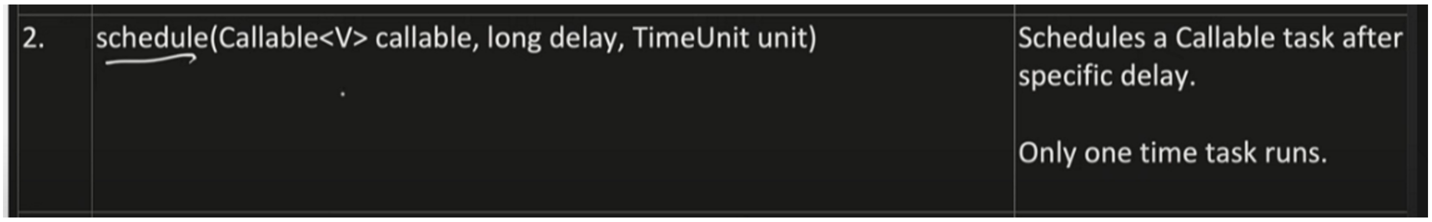

ScheduledThreadPoolExecutor : Helps to schedule the tasks
ThreadPoolExecutor
ScheduleThreadPoolExecutor

ScheduledThreadPoolExecutor has following methods



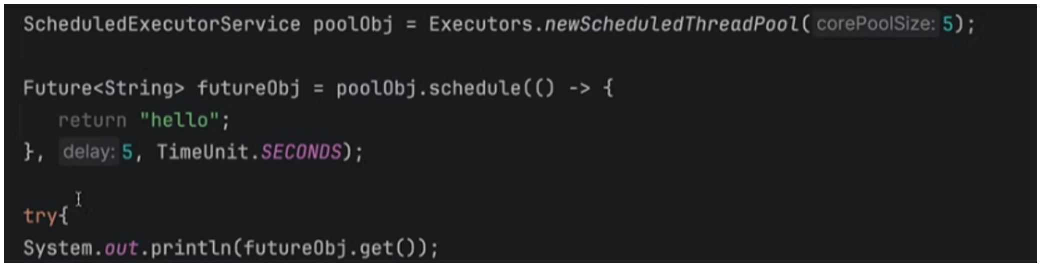
S.No.	Method Name	Description
1.	schedule(Runnable command, long delay, TimeUnit unit) schedule (() -> "hello", 3 , TimeUnit.SECOND)	Schedules a Runnable task after specific delay. Only one time task runs.

The runnable will be invoked after 3 second delay.



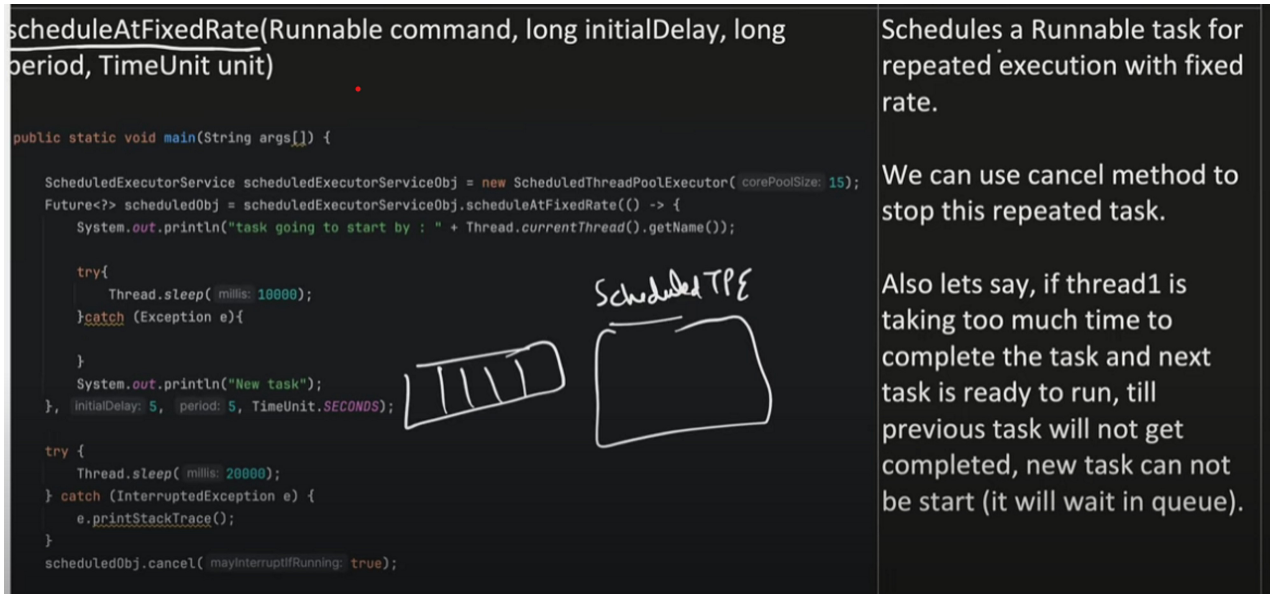
2.	schedule(Callable<V> callable, long delay, TimeUnit unit)	Schedules a Callable task after specific delay. Only one time task runs.

Here instead of runnable we have callable.



```
ScheduledExecutorService poolObj = Executors.newScheduledThreadPool( corePoolSize: 5);

Future<String> futureObj = poolObj.schedule(() -> {
    return "hello";
},  delay: 5, TimeUnit.SECONDS);

try{
System.out.println(futureObj.get());
```

But what if we want to run it more than once. For that we have a method scheduledAtFixedRate



scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)

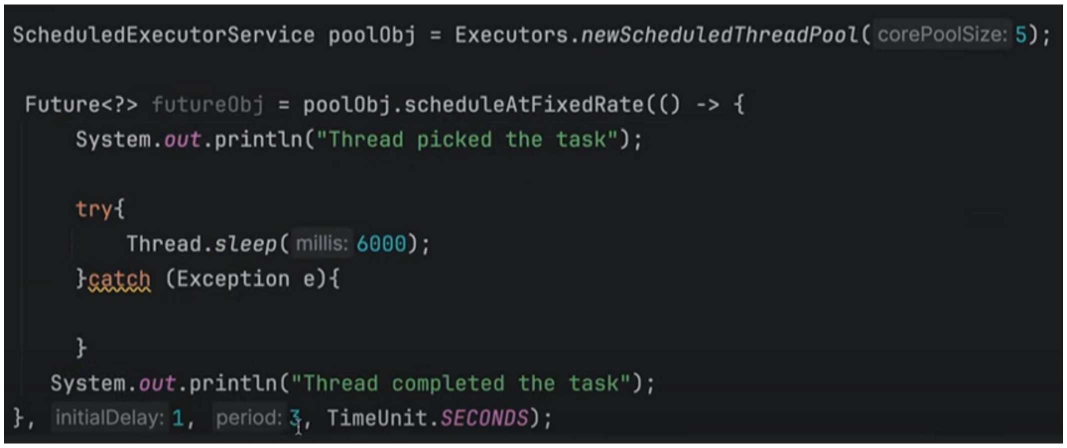
```
public static void main(String args[]) {

    ScheduledExecutorService scheduledExecutorServiceObj = new ScheduledThreadPoolExecutor( corePoolSize: 15);
    Future<?> scheduledObj = scheduledExecutorServiceObj.scheduleAtFixedRate(() -> {
        System.out.println("task going to start by : " + Thread.currentThread().getName());

        try{
            Thread.sleep( millis: 10000);
        }catch (Exception e){

        }
        System.out.println("New task");
    }, initialDelay: 5, period: 5, TimeUnit.SECONDS);

    try {
        Thread.sleep( millis: 20000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    scheduledObj.cancel( mayInterruptIfRunning: true);
```

Schedules a Runnable task for repeated execution with fixed rate.

We can use cancel method to stop this repeated task.

Also lets say, if thread1 is taking too much time to complete the task and next task is ready to run, till previous task will not get completed, new task can not be start (it will wait in queue).

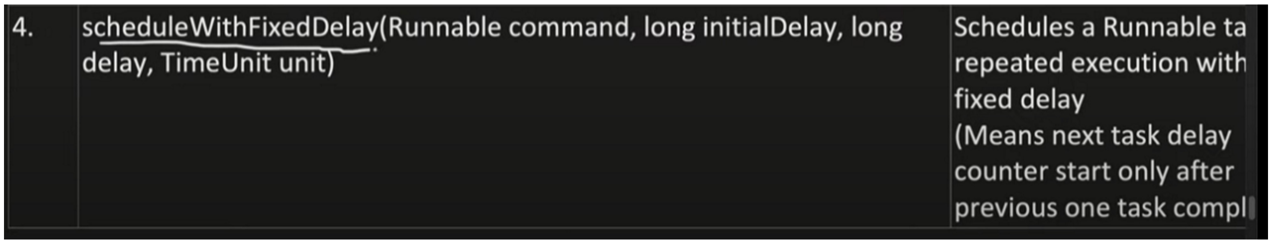
Period is after every 5 sec it will keep running
Lets see another example



```
ScheduledExecutorService poolObj = Executors.newScheduledThreadPool( corePoolSize: 5);

 Future<?> futureObj = poolObj.scheduleAtFixedRate(() -> {
     System.out.println("Thread picked the task");

     try{
         Thread.sleep( millis: 6000);
     }catch (Exception e){

     }
   System.out.println("Thread completed the task");
}, initialDelay: 1, period: 3, TimeUnit.SECONDS);
```

In above example, the other task that is after 4 sec will not get picked as first one itself has waiting time of 6 sec so the next iterations will be added in the queue and as soon as previous gets completed next task executes suddenly as 3 sec delay has already been overlapped.

There is another method called scheduledWithFixedDelay()



4.	scheduleWithFixedDelay(Runnable command, long initialDelay, long delay, TimeUnit unit)	Schedules a Runnable ta repeated execution with fixed delay (Means next task delay counter start only after previous one task compl

The difference here is once the previous gets completed only then the timer starts for the next. Like in previous example if we use scheduledWithFixedDelay then only after 6 sec the timer will wait for 3 more seconds then execute next task.